Carlo Batini
Fausto Giunchiglia
Paolo Giorgini
Massimo Mecella (Eds.)

LNCS 2172

Cooperative Information Systems

9th International Conference, CoopIS 2001
Trento, Italy, September 2001
Proceedings

Springer

Lecture Notes in Computer Science 2172
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Tokyo

Carlo Batini Fausto Giunchiglia
Paolo Giorgini Massimo Mecella (Eds.)

Cooperative Information Systems

9th International Conference, CoopIS 2001
Trento, Italy, September 5-7, 2001
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Carlo Batini
Massimo Mecella
University of Rome "La Sapienza", Department of Systems and Computer Science
Via Salaria 113, 00198 Roma, Italy
E-mail: batini@aipa.it; mecella@dis.uniroma1.it

Fausto Giunchiglia
University of Trento, Department of Information and Communication Technology
Via Sommarive 14, 38050 Povo, Trento, Italy
E-mail: fausto@ict.unitn.it

Paolo Giorgini
University of Trento, Department of Mathematics
Via Sommarive 14, 38050 Povo, Trento, Italy
E-mail: pgiorgini@cs.unitn.it

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Cooperative information systems : 9th international conference ; proceedings
/ CoopIS 2001, Trento, Italy, September 5 - 7, 2001. Carlo Batini ... (ed.).
- Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ;
Paris ; Tokyo : Springer, 2001
(Lecture notes in computer science ; Vol. 2172)
ISBN 3-540-42524-1

CR Subject Classification (1998): H.2.4, H.5, C.2.4, H.3, I.2.11

ISSN 0302-9743
ISBN 3-540-42524-1 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

http://www.springer.de

© Springer-Verlag Berlin Heidelberg 2001
Printed in Germany

Typesetting: Camera-ready by author, data conversion by DA-TeX Gerd Blumenstein
Printed on acid-free paper SPIN 10840428 06/3142 5 4 3 2 1 0

Preface

Cooperative Information Systems have emerged as a central concept in a variety of applications, projects, and systems in the new era of *e*-business. The conference at which the papers in this volume were presented was the ninth international conference on the topic of Cooperative Information Systems (CoopIS 2001), and was held in Trento, Italy on September 5-7, 2001.

Like the previous conferences, CoopIS 2001 has been remarkably successful in bringing together representatives of many different fields, spanning the entire range of effective web-based Cooperative Information Systems, and with interests ranging from industrial experience to original research concepts and results.

The 29 papers collected here out of the 79 ones that were submitted, demonstrate well the range of results achieved in several areas such as agent technologies, models and architectures, web systems, information integration, middleware technologies, federated and multi-database systems. The papers themselves, however, do not convey the lively excitement of the conference itself, and the continuing spirit of cooperation and communication across disciplines that has been the hallmark of these conferences.

We would especially like to thank our keynote speakers: Philip A. Bernstein (Microsoft Research, USA), Edward E. Cobb (BEA Systems, USA), and Maurizio Lenzerini (Università di Roma "La Sapienza", Italy) for providing a portrait of the best contemporary work in the field.

We would also like to thank the many people who made CoopIS 2001 possible. First of all, the program committee members and all the additional reviewers for all they did to ensure the high quality of accepted contributions. Second, Susanna Cavagna and Elisabetta Nones for doing all the necessary and hard work needed for the local organization and budget. Finally, Enrico Zaninotto, Dean of the Faculty of Economics, for letting us use the "Aula Magna".

July 2001

Carlo Batini
Fausto Giunchiglia
Paolo Giorgini
Massimo Mecella

CoopIS 2001 Conference Organization

CoopIS 2001 was jointly organized by the Università di Trento and Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", in cooperation with the International Foundation on Cooperative Information Systems (IFCIS). It was sponsored by the International Foundation on Cooperative Information Systems (IFCIS) and the Università di Trento.

Executive Committee

General Chair:	Fausto Giunchiglia Università di Trento, Italy
Program Chair:	Carlo Batini Università di Roma "La Sapienza", Italy Autorità per l'Informatica nella Pubblica Amministrazione, Roma, Italy
Organizing Chairs:	Susanna Cavagna, Elisabetta Nones Università di Trento, Italy
Publicity Chair:	Paolo Giorgini Università di Trento, Italy

Program Committee

Gustavo Alonso	ETH Zurich, Switzerland
Roberto Baldoni	Università di Roma “La Sapienza”, Italy
Carlo Batini (PC Chair)	Università di Roma “La Sapienza”, Italy and Autorità per l’Informatica nella Pubblica Amministrazione, Roma, Italy
Alexander Borgida	Rutgers University, USA
Paolo Bouquet	Università di Trento, Italy
Fabio Casati	HP Labs, USA
Paolo Ciancarini	Università di Bologna, Italy
Umesh Dayal	HP Labs, USA
Elisabetta Di Nitto	Politecnico di Milano, Italy
Ahmed Elmagarmid	Purdue University, USA
Alfonso Fuggetta	Politecnico di Milano, Italy
Mariagrazia Fugini	Politecnico di Milano, Italy
Dimitrios Georgakopoulos	Telcordia Technologies, USA
Paul Grefen	University of Twente, The Netherlands
Michael Huhns	University of South Carolina, USA
Christian Jensen	Aalborg University, Denmark
Dimitris Karagiannis	Universität Wien, Austria
John Krogstie	SINTEF Telecom and Informatics, Norway
Steve Laufmann	Qwest Communications, USA
Witold Litwin	University of Paris IX, France
Dennis McLeod	University of Southern California, USA
Massimo Mecella	Università di Roma “La Sapienza”, Italy
Zoran Milosevic	DSTC, Australia
Paolo Missier	Telcordia Technologies, USA
Michele Missikoff	IASI-CNR Roma, Italy
John Mylopoulos	University of Toronto, Canada
Shamkant Navathe	Georgia Institute of Technology, USA
Erich Neuhold	Darmstadt University of Technology, Germany
Oscar Nierstrasz	University of Berne, Switzerland
Tamer Özsu	University of Waterloo, Canada
Francois Pacull	Xeroc Research Centre Europe, France
Maurizio Panti	Università di Ancona, Italy
Mike Papazoglou	Tilburg University, The Netherlands
Barbara Pernici	Politecnico di Milano, Italy
Colette Rolland	Université de Paris 1 Pantheon Sorbonne, France
Marek Rusinkiewicz	Telcordia Technologies, USA
Heiko Schuldt	ETH Zurich, Switzerland
Ming-Chien Shan	HP Labs, USA
Amit Sheth	University of Georgia, USA
Keng Siau	University of Nebraska-Lincoln, USA

William Song	University of Hong Kong, Hong Kong
Stefano Spaccapietra	EPFL, Switzerland
Zahir Tari	Royal Melbourne Institute of Technology, Australia
Kyu-Young Whang	Korea Advanced Institute of Science and Technology, Korea

Additional Referees

S. Alfuraih
F. Anjum
K. Anyanwu
B. Arpinar
D. Arregui
D. Baker
K. Barker
M. Boehlen
A. Calì
D. Calvanese
J. Cardoso
A. Cichocki
M. Colajanni
P. Cremonesi
W. Derks
M.G. Elfeky
J. Erickson
M.M. Fokkinga
C. Francalanci
M.M. Hammad
N. He
E.K. Hong
S. Kulkarni
D. Lee
C.H. Lin
C. Marchetti
M. Mesiti
Y. Ndiaye
A.H.H. Ngu
M.E. Poleggi
T. Risse
A. Salminen
M. Scannapieco
H.W. Shin
G. Slivinskas
J. Steel
F. Taglino
N. Tryfona
P. van Eck
A. Virgillito
L.L. Yan
X.F. Wang
W. Wang
J. Willamowski
F. Zito

Table of Contents

Generic Model Management: A Database Infrastructure for Schema Manipulation

Philip A. Bernstein

Microsoft Corporation, One Microsoft Way
Redmond, WA 98052-6399 USA
philbe@microsoft.com

Abstract. This paper summarizes the model management approach to generic schema management. The goal is to raise the level of abstraction of data manipulation for programmers of design tools and other model-driven applications. We explain the main concepts of model management, report on some recent progress on model matching algorithms, and sketch a category-theoretic approach to a formal semantics for model management.

1 Introduction

Many of the problems of cooperative information systems involve the design, integration, and maintenance of complex application artifacts: application programs, databases, web sites, workflow scripts, formatted messages, user interfaces, etc. Engineers who perform this work use tools to manipulate formal descriptions, or *models*, of these artifacts: object diagrams, database schemas, web site layouts, control flow diagrams, XML schemas, form definitions, etc. The tools are usually *model-driven*, in the sense that changes to the tool can be accomplished by updating the tool's meta-model. For example, an entity-relationship design tool may allow a user to add fields to the types Entity, Attribute, and Relationship, and to add a function that uses those fields to generate SQL DDL. This allows the user to customize the tool for database systems that have special DDL capabilities not anticipated by the tool designer. Often, the run-time application itself is model-driven. For example, a message-translator for business-to-business e-commerce might be built as an interpreter of mappings that describe transformations between types of messages. Changes to the message-translator can be accomplished simply by modifying the mapping.

Model-driven tools and applications need a persistent representation of the models that they manipulate. The database field's last big push to meet the special needs of these types of design applications was the development of object-oriented database systems (OODBs) starting in the mid-to-late 1980s. Yet today, most model-driven applications are built using relational databases and files. This is rather odd, given that OODBs not only have the usual database amenities, such as a query language and transactions, but also offer several

© Springer-Verlag Berlin Heidelberg 2001

valuable benefits over relational databases: a rich type system in which to describe models, smooth integration with object-oriented programming languages, a versioning facility, and dynamically extensible types. These features are surely helpful to developers of model-driven applications, among others, yet customer acceptance of OODBs has been disappointing. The reason usually cited is the difficulty of displacing a dominant technology like SQL database systems by a less mature and robust one like OODBs. This may not, however, be the whole story.

In our opinion, what is missing from today's persistent storage systems are features that offer an order-of-magnitude productivity gain for programmers of model-driven applications over existing relational and OO databases, much like relational database systems achieved over their predecessors for business-oriented applications. Such an order-of-magnitude gain requires a leap of abstraction in the programming interface. The opportunity to achieve this gain lies in the fact that most model-driven applications include a lot of record-at-a-time or object-at-a-time navigational code, even when a query language like SQL or OQL is available. To avoid this navigational programming, we need a higher-level data model, one with higher-level operations on higher-level objects that would greatly reduce the amount of navigational code required by model-driven applications. Our proposal for this higher level data model is summarized in the next section.

2 Models and Mappings

In [4], we proposed an algebra that manipulates models and mappings, each of which connects the elements of two models. The algebra is meant to be generic, in the sense that it can be applied to models (i.e., schemas) expressed in many different data models. Some of the key operations are:

- Match - which takes two models as input and returns a mapping between them
- Compose - which takes two mappings as input and returns their composition
- Merge - which takes two models and a mapping between them as input and returns a model that is the merge of the two models (using the mapping to guide the merge)
- Set operations on models - union, intersection, difference
- Project and Select on models - comparable to relational algebra

To see how these operations might be used, consider the problem of populating a data warehouse. Suppose we have a model S_1 of a data source and a mapping map_{1W} from S_1 to a model W of a data warehouse. Now we are given a model S_2 of a second data source (see Figure 1). We can integrate S_2 into the data warehouse as follows: (1) Match S_2 and S_1, yielding a mapping map_{21}; (2) Compose map_{21} with map_{1W}, to produce a mapping map_{2W} from S_2 to W. Step (1) characterizes those parts of S_2 that are the same as S_1. Step (2) reuses map_{1W} by applying it to those parts of S_2 that are the same as S_1. In [5], we described a detailed version of this scenario and others like it, to show how to use the model management algebra to solve some practical data warehousing problems.

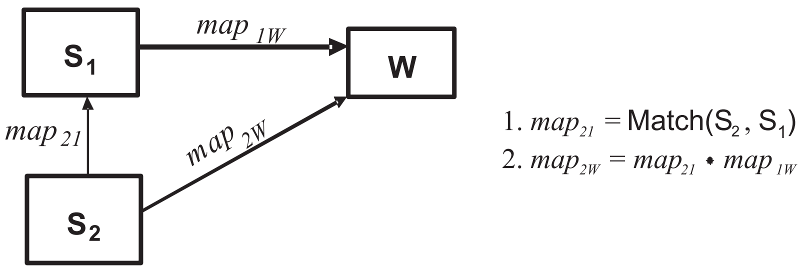


Fig. 1. A Simple Data Warehouse Scenario Using Model Management

Even if one agrees that the above algebra is a plausible candidate for raising the level of abstraction of meta-data management, there is still the question of defining a simple and precise semantics for generic schema management operations. A panel on this question took place at VLDB 2000 [3]. On the whole, the panelists found it worthwhile to pursue the goal of generic schema management. However, there was some disagreement on how difficult it will be to define the semantics of operations that can address the complexity of real-world data.

The examples in [5] were one step toward defining and validating an operational semantics to handle this comlexity. More recently, we have made progress on two related fronts. They are described in the next two sections.

3 Schema Matching

To bootstrap the development of a model management implementation, we need mappings. Hence, the first operation of interest is the one that generates mappings, namely, Match.

It is implausible that one can develop a fully automatic implementation of Match. The main reason is that most models are incomplete. That is, they have a lot of semantics that is known or discoverable by human designers but is not formally expressed in the model. Despite this inherent incompleteness, there is still quite a lot of information that a match algorithm can use to produce a draft mapping, for review by a human designer. Such information includes the names of objects, their data types, their overall structure, example instances, and auxiliary information about the domain of interest, such as a thesaurus and glossary of acronyms.

There is a substantial research literature on match algorithms that exploit this kind of information. We surveyed them in [9], where we developed a taxonomy of approaches. Using this taxonomy for guidance, we integrated many of these approaches into a hybrid algorithm, called Cupid. Cupid uses properties of individual elements, linguistic information of names, structural similarities, key and referential constraints, and context-dependent matching. The algorithm is described in [6], along with an experimental comparison to two earlier implementations, DIKE [8] and MOMIS [2], some of whose ideas were incorporated into Cupid. While a truly robust solution is not around the corner, results so far have been promising.

We have found that our implementation of Match is immediately useful to some tools, even without implementations of the other model management operations. For example, consider a graphical tool for defining a mapping between XML message types. By incorporating Match, the tool can offer a user a draft mapping, which the user can then review and refine. We expect that an implementation of other operations will be similarly useful outside of a complete model management implementation.

4 Formal Semantics

To achieve the goal of genericity, we need a semantics for model management that is independent of data model. One approach is to define a rich data model that includes all of the popular constructs present in data models of interest. This would be an extended entity-relationship (EER) model of some kind, which includes an is-a hierarchy, complex types, and the usual kinds of built-in constraints, such as keys and referential integrity. This is the approach used in the Cupid prototype. Models expressed in some other data model, such as SQL DDL or XML schema definitions (XSD [10]), are imported into the native Cupid data model before they are matched. The resulting mapping is then exported back into the data model of interest, such as one that can be displayed in a design tool (e.g., for SQL or XSD). That way, Cupid can support many different data models and, hence, many different applications.

It would be desirable to have a carefully developed mathematical semantics that shows that the EER data model used by Cupid, or one like it, is generic. One way to go about this would be to demonstrate a translation of each data model of interest into the chosen EER model and analyze properties of that translation, such as how much semantics is preserved and lost. This approach is likely to be enlightening, but it is still on the docket.

A second approach is to analyze models and mappings in the abstract, using category theory [7]. A category is an abstract mathematical structure consisting of a set of uninterpreted objects and morphisms (i.e. transformations) between them. To apply it to models and mappings, one represents the well formed schemas of a data model as a category, whose internal structure is uninterpreted. Models are objects of a schema category, and mappings between models are morphisms between objects of a schema category. A mapping between two categories is called a functor. A functor can be used to provide an interpretation of schemas; the functor maps each schema in a schema category to a category of instances, which are the set of all databases that conform to the schema. One can then use some standard constructions in category theory to study the effects of combining mappings in certain ways. For example, consider two mappings that integrate each of two data source schemas into a target schema. This construction resembles the category-theoretic concept of ôpushout,ö which in this case is a minimal target schema that is, in effect, a subschema of any other integration of the two data sources.

We have used this categorical approach to study the effect of schema integration on integrity constraints [1]. The concept of an expression language for constraints is defined by an uninterpreted functor that maps each schema to the set of well-formed formulas over that schema (i.e., constraints that use the schema's objects as terms in the formulas). The set of well-formed formulas is just a set of sentences. At the level of category theory, they're uninterpreted. This allows us to talk about constraint satisfaction in general, independent of the logic that is chosen to express the constraints. Using this abstract model of constraints, we can characterize the properties of a constraint preserving integration of data source schemas and their integrity constraints into a target schema with its integrity constraints.

Category theory gives the "shape" of the model management theory. That is, it explains the main theorems to be proved for any given data model. To make that theory useful, one has to apply its results to specific categories of interest, such as relational, OO, or XML schemas plus an associated constraint logic, and to specific types of schema transformations. That application of the theory uses the additional structure of the schema types and constraints of interest to drive through the proofs of those theorems. We did this for a simple OO schema language and simple transformations. It would be valuable to extend this to other data models and more complex kinds of transformations.

This study of schema integration serves as a template for using the categorical approach to study other schema mapping problems, such as database evolution and migration, and to study the semantics of transformations that correspond to model management operations. It also demonstrates a tangible benefit of the categorical approach, namely a generic formal study of constraints in schema mapping, the first such treatment we know of.

5 Conclusion

We have briefly reviewed the model management approach to generic schema management and summarized some recent progress in the areas of model matching and formal semantics. These are steps along what we expect to be a very long research path. Other topics worth investigating are the semantics and implementation of Merge and Compose, properties of translations from standard data model into a generic EER one, and the development of detailed examples that apply model management to practical problems.

Acknowledgments

The author is very grateful to the following researchers whose work is summarized in this paper: Suad Alagic, Alon Halevy, Jayant Madhavan, Rachel Pottinger, and Erhard Rahm.

References

1. Alagic, S. and P. A. Bernstein: "A Model Theory for Generic Schema Management," submitted for publication. 5
2. Bergamaschi, S., S. Castano, and M. Vincini: "Semantic Integration of Semistructured and Structured Data Sources." SIGMOD Record 28,1 (Mar. 1999), 54-59. 4
3. Bernstein, P. A.: "Panel: Is Generic Metadata Management Feasible?" Proc. VLDB 2000, pp. 660-662. Panelists' slides available at http://www.research.microsoft.com/ philbe. 3
4. Bernstein, P. A., Halevy, A., and R. A. Pottinger: "A Vision for Management of Complex Models." ACM SIGMOD Record 29, 4 (Dec. 2000). 2
5. Bernstein, P. A. and E. Rahm: "Data Warehouse Scenarios for Model Management," Conceptual Modelling - ER2000, LNCS1920, Springer-Verlag, pp. 1-15. 3
6. Madhavan, J., P. A. Bernstein, and E. Rahm: "Generic Schema Matching Using Cupid," VLDB 2001, to appear. 4
7. Mac Lane, S.: Categories for a Working Mathematician, Springer, 1998. 4
8. Palopoli, L. G. Terracina, and D. Ursino: "The System DIKE: Towards the Semi-Automatic Synthesis of Cooperative Information Systems and Data Warehouses." ADBIS-DASFAA 2000, Matfyzpress, 108-117. 4
9. Rahm, E., and P. A. Bernstein: "On Matching Schemas Automatically," Microsoft Research Technical Report MSR-TR-2001-17, Feb. 2001. 3
10. W3C: XML Schema, http://www.w3c.org/XML/schema, 2001. 4

The Evolution of Distributed Component Architectures

Edward E. Cobb

BEA Systems, Inc.

Abstract. The growth of the Internet has coincided with the adoption of component system architectures. But early component architectures were built on assumptions that do not translate well when multiple business enterprises are involved. Nevertheless, these technologies have become widely adopted within the enterprise and form the basis for the new architectures that are emerging for business to business solutions. This paper looks at the evolution of component architectures in the past decade and tries to draw some conclusions about how these systems will evolve in the next several years.

1 Introduction

Business today is faced with tremendous competitive pressure to ensure survival. The *information revolution*, powered by the Internet, is changing the fundamentals of commerce which have prevail since the industrial revolution. For the first time in history, the customer, not the supplier, is in control of the value chain and that customer is demanding better and more personalized services. Expectations for services that will be "dial tone" quality --- always there and without noticeable delays, are likely to be the defining characteristic of the way business will be done in the 21st century. For businesses unable to adapt, there are competitors waiting --- ready and willing to do so, just a mouse-click away.

Companies like Dell and WalMart are pioneering this transformation, and they are taking market share from the established leaders in their respective markets and these companies are scrambling to avoid being left behind. With today's corporation defined by its information systems, this has placed a premium on applications which can rapidly adapt to the forces of change. *Component technology* has emerged as the preferred solution for building these adaptive applications. Component architectures allow applications to be developed faster, enabling businesses to act more rapidly to gain and maintain competitive advantage.

2 Component Technology

Component technology is to software what manufacturing is to hardware. In hardware, the initial design is developed by highly-skilled engineers at a premium cost and takes considerable time to produce. Once it has been successfully proven however, it can be mass produced for a fraction of its original cost and be used in a multiplicity of products besides than the one it was originally designed for. Components promise the same kind of leverage for software development, enabling business applications to be constructed

© Springer-Verlag Berlin Heidelberg 2001

from pre-built parts delivering higher quality solutions more rapidly using less skilled resources.

2.1 Components and Objects

The first software technology to address the productivity challenges of enterprise software development was *object technology*. Although object technology has its genesis in the late 1960s with languages like Simula, it entered the software mainstream in the late 1980s. The claims of its advocates were directly analogous to hardware development. Software objects would be fine-grained "chunks" of reusable code which could be put together in a variety of different ways to produce different applications. These objects would either be produced by the more experienced programmers in the customers IT organization or, in some future economy, would be available "off the shelf" from a variety of suppliers in the same way that computer system manufacturers like Compaq and HP buy chipsets from semiconductor manufacturers.

2.1.1 Objects on the Desktop

The first commercial success with object technology was on the desktop of the personal computer and workstation where it was used to build graphical end-user interfaces. This environment provided a perfect proving ground to apply the principles of object technology. The principal user of this new computing platform was the consumer, not the IT professional. Apple was the first to exploit this technology with its Macintosh operating system. The Macintosh was unique in offering only a graphical interface for end users. It had no command line interfaces like Unix®, PC DOS, and every operating system before it and its intuitiveness and simplicity made it an immediate success in the consumer marketplace.

The highly interactive end-user paradigm of the desktop demanded a systems designed with fine grain "chunks" of software to respond to mouse clicks and keyboard inputs. Both Microsoft and IBM responded to the success of the Macintosh by designing the next generation PC operating system, initially together but ultimately as competitive offerings. Microsoft's Windows™ would ultimately prevail with its Component Object Model (COM™) technology [COM] and components, in the form of fine grain objects, would become the *de facto* standard for desktop development.

An important characteristic of the desktop programming paradigms is the *tightly-coupled* nature of the interactions between the various system objects providing the desktop functionality. A tightly-coupled architecture relies on a large amount of shared context. Because all the desktop objects executed on the same operating system, each "chunk" of software had a high degree of dependency on other "chunks" that it interacted with, not a surprise since they were all dealing with the same physical display. This design characteristic would carry over to early attempts at bringing the benefits of object technology to the server and the distributed systems environment.

2.1.2 Distributed Objects - CORBA and DCOM

In 1989, the Object Management Group (OMG) was founded with a charter to promote a single object-oriented applications integration environment based on appropriate industry standards.[1] Over the years the organization has developed a variety of interoperability specifications based on a standard communication mechanism (Internet Interoperability Protocol - IIOP) and a mapping of that protocol to a variety of programming languages from C to Java™ and almost everything in between. These specifications are collectively known as the Common Object Request Broker Architecture (CORBA™).

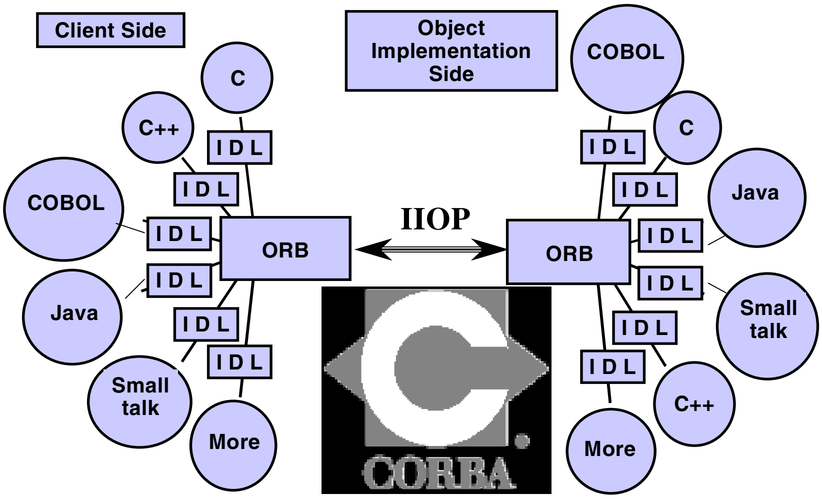


Figure 1 Distributed Objects Using CORBA

In the CORBA object model [CORB] communicating objects share a contract, defined in Interface Definition Language (IDL), which specifies the operations and parameters of the client's interaction with a remote object. The IDL representation of the contract is then independently mapped to the programming language used by the client and the remote object, which need not be the same. The *Object Request Broker* (ORB) is a software bus which allows the client to make calls locally, packages the calls for transmission and sends the call across the wire in a standard format. On the server side, the ORB unpackages the client's request and then makes a local call on the target object. The net effect is a *remote procedure call* (RPC).

Procedure calls, which are an artifact of tightly-coupled architectures, are very natural programming paradigms and almost every programming language enshrines the concept in the language itself. Providing transparent distribution of a procedure call by the ORB made it relatively easy for programmers to start using distributed objects. This same concept was also arrived at by Microsoft in developing a distributed version of its COM object model (DCOM™) to enable COM objects on multiple Windows systems to communicate.

2.2 Client/Server and the Rise of the Web

The personal computer introduced a new wave of computing, the *client-server* revolution. Low cost computer hardware together with the inability of central IT organizations

1. OMG Bylaws section 1.2, January 1999 revision, http://www.omg.org

to respond to the needs of end-user departments, encouraged these departments to build solutions themselves. With the client-server model, business logic resides in the client's computer. The server (in most cases also a personal computer class machine connected via a local area network) became the means to share peripherals (e.g. printers and disks) and services (e.g. files and databases) used by multiple applications. This *two-tier* architecture would become the dominant programming paradigm of that era.

With new development being done on the client, the use of object technology in the form of Microsoft's COM grew. Visual Basic [MVSB], an outgrowth of the venerable BASIC programming language, became the language of choice for the business programmer. C++ replaced C for the power programmer and the existing procedural languages, although ported to the PC, never gained much traction.

By the early 1990s, two trends emerged which would ultimately mean the end of the client-server era:

- client-server solutions proved difficult to manage and even more difficult to scale
- The world wide web and the browser appeared on the scene

Both of these trends were addressed by very similar solutions -- the *Distributed TP Monitor* and the *Application Server*.

2.3 Distributed TP Monitors and Application Servers

The distributed TP monitor was a *n-tier* solution, much easier to manage, and promising virtually unlimited scalability. Tuxedo, originally developed by Bell laboratories as a Unix TP monitor, was extended to support distributed systems.[1] IBM introduced a distributed version of its popular mainframe CICS TP monitor based on the ENCINA[2] technology developed by its Transarc subsidiary. Tuxedo was spun-off to the Unix System Labs as part of the break-up of AT&T and was subsequently sold to Novell before being acquired in 1995 by BEA Systems, who would make it a commercial success. Distributed CICS had more limited success, but both would be engulfed by the growth of the web and the emergence of the application server. Both Tuxedo and CICS had an abstraction that was optimized for distribution. Tuxedo called them *services* and CICS called them *transactions*.

The application server was a new breed of middleware, based on the web browser as a thin client. Browsers provided a simple user interface on the PC, without client programming, and used the Internet to send requests for data stored elsewhere. Initially all of that data was static content in the form of graphical pages, but gradually logic was introduced, making it possible to use the Internet as a front-end to both corporate data and applications. When Java arrived on the scene in the early 1990s, demand for application servers exploded and a new market segment was firmly established. Initially, numerous commercial offerings flourished, but lately consolidation has been occurring

1. TUXEDO actually stands for Transactions on UNIX, extended for distributed operations.
2. ENCINA stands for Enterprise Computing in a New Age

with BEA's WebLogic Server™ [WEBL] and IBM's Websphere™ [WEBS] battling for market supremacy.

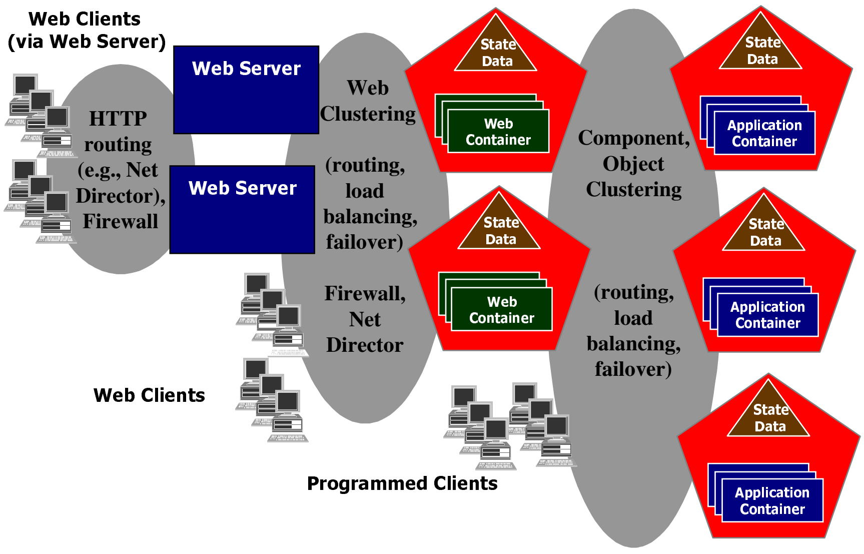


Figure 2 The Application Server Architecture

The application server re-established the server as the preferred application deployment platform and re-invigorated the quest for a technology that would make server development more productive.

2.4 From Objects to Components

Distributed objects were an important step in the evolution of component-based software but were insufficient in themselves to realize the promised benefits. Distributed objects systems were based on an ORB messaging structure and were lacking in two major areas:

- object systems were often designed with small, fine-grain objects which were not able to be distributed efficiently
- the services necessary to implement distributed objects efficiently were lacking in the available ORB products

To make component-based development a reality and to start seeing its potential benefits, both of these problems needed to be addressed.

2.4.1 Distribution Granularity

Although communication technology has improved significantly in the last decade, remote access is (and will continue to be) more expensive in both response time and processor utilization. **Applications must be properly designed to use distribution effectively**. Applications that overuse distribution will typically suffer poor performance and in almost all cases will not scale. An architecture which provides local-remote transparency (like both CORBA and DCOM) makes it too easy for programmers to ignore the effects of distribution on their designs until it is too late. As a result, many application systems were designed with objects that were called too frequently across network connections.

Object oriented designed methodologies [OOAD, OOSE] encouraged functional decomposition to very fine-grain objects even when all of these objects did not need to be remotely accessed. The need for something "bigger than an object" to design efficient distributed object systems was apparent. Ultimately this would become the component.

2.4.2 The Need for a Server-Side Platform

The second shortcoming of distributed object architectures such as CORBA was the absense of a *platform* for the object implementation. In a single system, that platform is the operating system. Operating systems provide a variety of commonly used services as well as the invisible qualities of reliability, performance, and scale which support efficient execution. But operating systems are designed to manage a single machine environment, not a network. An operating system, which itself was distributed, was necessary to provide the platform needed by a distributed object system.

Although CORBA is rich in system service specifications, many were not widely implemented by commercial products. The most popular commercial ORBs at the time, Iona's Orbix [ORBX] and Borland's Visibroker [VISI], provided only basic CORBA functionality on top of a simple RPC mechanism. Microsoft's DCOM had a very similar design. Both would discover this approach was not the answer to providing a highly scalable distributed component architecture.

So when the programmer came to develop the implementation of his business object, he also had to build many elements of the platform infrastructure to support it. Standard services like transactions, security, and persistence were required almost universally and they were often implemented as part of the application. Implementing these services in a reliable and scalable fashion requires the design skills more typical of middleware software providers than the business programmers found in the typical customer's IT shop.

The solution to this problem was to be found in a form of middleware that has been around since the late 1960's, viz. the *TP monitor*. Although often portrayed by the new object-oriented evangelists as "old technology," it had precisely the implementation qualities needed for scalable, reliable objects as well as off-the-shelf implementations of common services. In an Object Magazine article [TPMO] I authored in 1995 I speculated:

> The marriage of ORB and TP monitor technologies will create a new infrastructure, exploiting the strengths of TP monitors, while providing a robust application development environment based on reusable objects.

BEA's M3 product, based on the Tuxedo TP monitor, was the first to integrate CORBA technology with transaction processing middleware. Microsoft would take a similar approach with its Microsoft Transaction Server (MTS) for COM objects. The transaction infrastructure of the TP monitor would become the requisite platform to make server-side components a reality and would be provided by the emerging category of system software, the *application server*.

2.5 Enterprise Java Beans

EJB [EJB] is one of several technologies which make up Sun's Java 2 Enterprise Edition™ (J2EE) specification. The J2EE specification [J2EE] defines a common multi-system, vendor neutral platform for building applications using the Java programming language. EJB is a programming specification which is implemented by multiple commercial middleware vendors, so that application programmers can more easily deal with infrastructure issues in developing server programs. EJB defines a *container model* that the EJB programmer can be assured is the same, regardless of which EJB-compliant product the application is deployed on.

EJB defines two categories of components (beans, in EJB terminology): **session** and **entity**. Session beans provide the level of abstraction required to effectively use distribution without inordinate network cost. Session beans actually have two flavors: *stateless* and *stateful*. Stateless Session Beans are component abstractions functionally equivalent to a service in BEA's Tuxedo™ [TUX] or a transaction in IBM's CICS™ [CICS]. Stateful session beans have conversational state for the lifetime of the bean, similar to a conversational service (or transaction). Session beans are associated with a client for the duration of the client's session and are not shared.

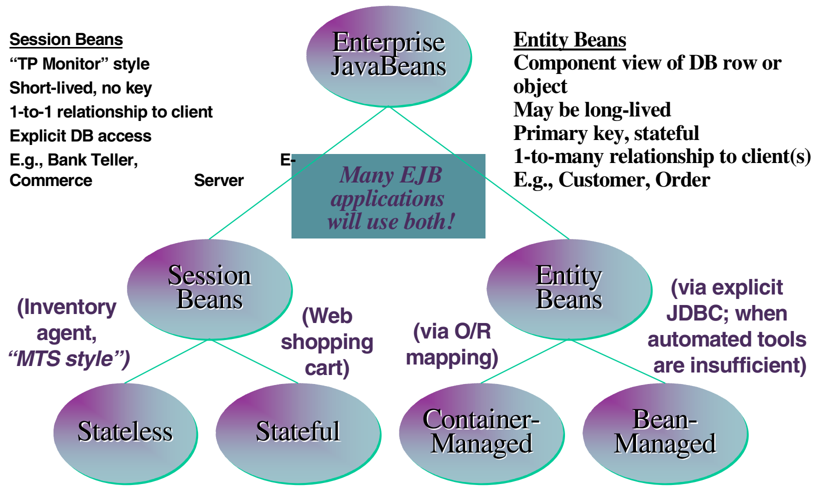


Figure 3 Enterprise Java Beans Taxonomy

Entity beans, on the other hand, are intended to represent persistent data that is typically stored in a database. In the EJB model, entity beans are identified by a *primary key* and can be shared among multiple clients. Although the architecture permits remote access, good systems design dictate that you expose only coarse-grain objects (session beans) rather than fine-grain objects (entity beans) across the network. This reduces the numbers of messages and encourages transparency of database design (strongly related to entity beans). The state associated with entity beans can be automatically managed by the container (**container-managed persistence**) or managed by the bean itself (**bean-managed persistence**). A single application may use beans of both categories.

The EJB programming model is, in reality, a set of design patterns formalized by the EJB component architecture. EJB was designed by experts from the leading transaction processing vendors and these design patterns are the direct result of their thirty plus years of experience in designing and deploying transaction processing systems solutions. The EJB container is the platform abstraction provided to the application programmer. By standardizing commonly used services and successful design patterns, the application programmer is free to concentrate on the logic associated with the business process.

3 Business to Business Architectures

Like all of its predecessors, the first versions of EJB were based on a tightly-coupled architecture. EJB Version 1 [EJB1] supports remote invocation using Java Remote Method Invocation (RMI), a simple RPC mechanism. J2EE, however, is more than the EJB component model. It is a complete set of platform services for EJB applications including the Java Messaging Service (JMS) [JMS]. JMS makes it possible to build loosely-coupled architectures on the J2EE platform. EJB Version 2 [EJB2], to be available in 2001, uses JMS to add support for *message-driven* beans extending the EJB component model to support both tightly-coupled and loosely-coupled architectures.

J2EE applications can rely on a shared infrastructure of common services and transparent support for reliable, scalable execution. These properties are provided by the application server vendor and customers have the ability to choose between multiple vendors who compete on the quality of their individual implementations. Many companies have embraced this technology and deployed it widely within their own enterprise. But taking the next step and using it to interact with suppliers and customers is not straightforward. And the reasons are not technical.

3.1 What's Different about Inter-Business Communication

Consider the state transition diagram in **Figure 4** Booking a Business Trip (step S1).[1] An airline reservation (step S3), a hotel (steps S4 or S6), and a rental car (steps S5 or S7) are all required (step S9), and need be coordinated with each other or the trip will not be taken (step S8). Since the inventory for each of these is maintained by their respective owners, an agent (e.g. the travel agent) must execute the correct sequence of transactions to ensure a consistent outcome, viz.

- a seat is reserved on a flight to your intended destination,
- your rental car is waiting at your destination airport, and
- your hotel room is reserved for the evening of your arrival at your intended destination.

1. This example is taken from an earlier paper [OTTP] published in the VLDB journal in 1997.

If any of the steps fail, the sequence must be restarted and alternate actions must be taken. These alternate actions are referred to as *compensations* in the literature [SAGA]. For example, if you are headed to New York and the last flight to JFK is sold out, you might be willing to fly into Newark airport, but a rental car waiting at JFK isn't going to be of much help, so you need to cancel the JFK rental car reservation and book a new one at Newark airport.

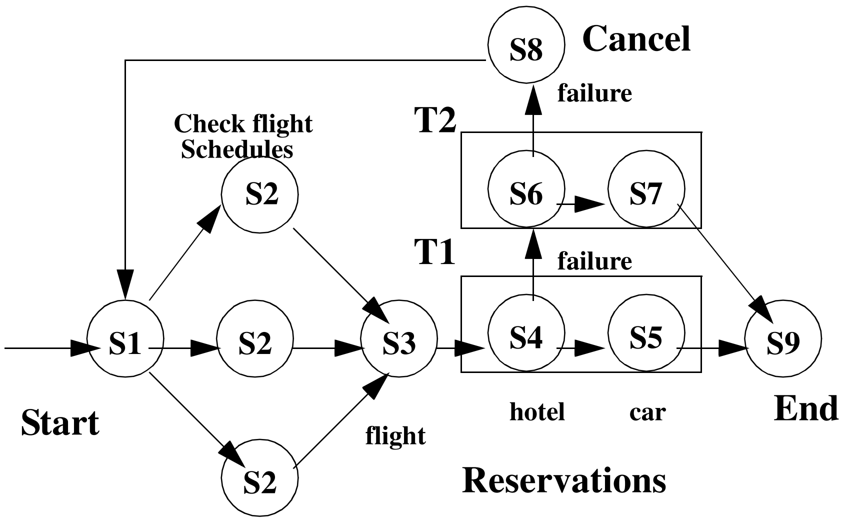


Figure 4 Booking a Business Trip

An important characteristic of this business transaction, which involves four separate businesses is that each step is executed by one and only one of the suppliers and each supplier has no knowledge of any of the others except as provided by the travel agent, who orchestrates the business transaction. The steps are not only disjoint in time, but also in their ability to share data. As a result, a loosely-coupled architecture must be employed.

Since this same example is often used to explain transactions and the two-phase commit protocol (2PC), one may question why it can't be used for this example. Transactions are one of several properties that are shareable with tightly-coupled architectures (security is another example) and sharing data would be necessary to make this process atomic using traditional transaction technology [TPCT].

To implement this example in the real world, a loosely-coupled architecture must be chosen. With loosely-coupled architectures, the only context that needs to be shared is the actual data being exchanged. And that data is part of the legal contract between the companies involved in the business process of booking a trip.

3.2 Electronic Data Interchange

Many larger enterprises have used Electronic Data Interchange (EDI) as a standard mechanism for exchanging information with their partners and suppliers. but EDI solutions are expensive, requiring private networks and complex software. This is **not** a feasible choice for smaller businesses.

Fortunately, today even the smallest of businesses have access to the Internet and browsers and e-Mail provide ubiquitous connectivity. What's needed (and there are lots of industry standards groups working in these areas) is the equivalent of EDI data format standards based on web protocols. XML (eXtensible Markup Language) has emerged as the *lingua franca* for exchanging business information over the web [XML].

Loosely-coupled architectures do not eliminate the need for an application platform. Platform services serve the same purpose as they do for applications using tightly-coupled architecture. The difference is the **scope** of the shared context.

- With tightly-coupled architectures, context is shared across the entire network of cooperating systems.
- With loosely-coupled architectures, context is not shared

A *domain* is a collection of systems which are administered as a unit. In almost every case, a domain will **not** cross enterprise boundaries. In fact, there may be several of these domains within a single enterprise, or, at the other extreme, a enterprise may consist of only a single domain. Sharing context within a domain is quite simple, since the systems are homogeneous (e.g. all J2EE platforms). Sharing context across domain boundaries, when both use tightly-coupled architectures (e.g. communicating between a J2EE application server and a TP monitor environment), requires transforming shared context at the boundary. If the domains are not tightly-coupled, there is no shared context to be transformed. Within a single domain, some applications may choose to use tightly-coupled architectures and some loosely-coupled.

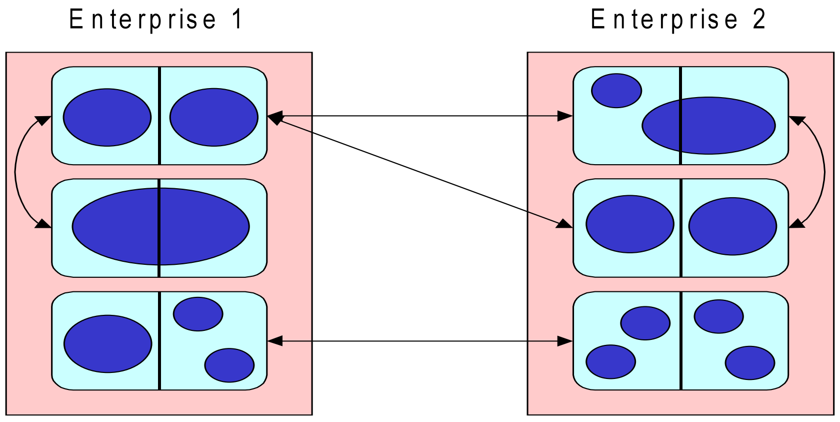


Figure 5 Tightly-coupled and Loosely-coupled domains

J2EE is an example of a platform supporting both types of domains. JMS and the message-driven beans of EJB Version 2 are the key technologies. But unlike tightly-coupled architectures, other platform offerings are certainly possible. Microsoft's .NET initiative [MNET] is targeted specifically at providing a new loosely-coupled architecture designed for exchanging information over the Internet.

3.3 Web Services

Web Services is used to denote an application model that describes an architecture for implementing shareable services on the Web. The web services model is an example of a loosely-coupled architecture. With the web services model:

- interactions are between services, rather than between a person using a browser and the service
- services can be described, published, discovered and invoked dynamically over the web by other services
- the service structure is peer-to-peer (P2P), in contrast to the client server structure of today's browser to service paradigm.
- like all loosely-coupled architectures, the web services model is language-independent and platform-agnostic.

The web services model makes new web applications feasible, e.g. intelligent agents can discover new trading partners, market places can be created for auctioning of goods and services, and business deals can be transacted among a multitude of participating partners rather than the traditional one-on-one relationships. Some examples of web services include:

- News feeds
- Airline schedules
- Stock quotes
- Credit checks

The web services architecture includes three element types:

- a service provider, who publishes the availability and the nature of its services in a registry
- a service broker who supports, via its *registry*, publication of the location of services offered by providers
- a service requestor, who finds services of interest via the service broker and once found, binds to the services via the service provider.

The elements of a web services system are illustrated in **Figure 6** Web Services Architecture:

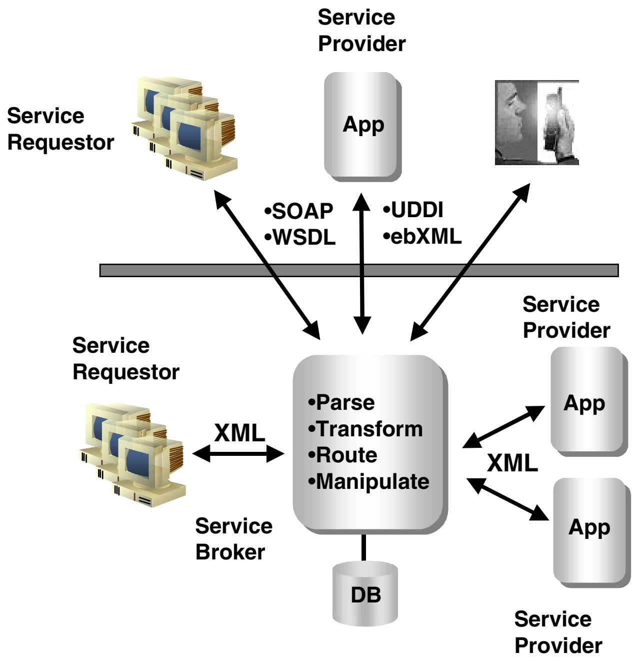


Figure 6 Web Services Architecture

One of the goals of web services is to realize a distributed architecture across the web that is platform independent. Hence, compliance with well-established and generally accepted standards is essential. One of the most important is XML, which acts as the universal data format for web services. XML and XML-based standards such as XSL Transformations (XSLT), provide translation capabilities between the new XML data formats and data formats used by a variety of other platforms, including TP monitors like Tuxedo, application systems like SAP R3, as well as the J2EE application server and Microsoft's .NET.

As of May 2001, a number of concurrent initiatives are under way to create the required standards. Two of the most important are Universal Description, Discovery, Integration (UDDI) and Electronic Business for XML (ebXML).

UDDI [UDDI] is a consortium of vendors focused on directory services whose goal is to define a service registry for the web services architecture. Announced in September 2000 by Microsoft, IBM and Ariba, it now includes almost every major software vendor as well as a large number of business users. The initiative has three principal elements:

- The UDDI registries, which act as a directory of available services and service providers.
- SOAP (Simple Object Access Protocol), which is an XML-based protocol for service invocation.
- WSDL (Web Services Description Language), which is an XML vocabulary to describe operational information about the service.

ebXML [BXML] is a joint initiative by the Organization of the Advancement of Structured Information Standards (OASIS) [OASI] and the Center for Trade Facilitation and Electronic Business of the United Nations (UN/CEFACT). Its aim is to create an open XML-based infrastructure for business-to-business communication. It lists over 1000 active participants and over 850000 [EBPR] companies committed to use the standards, which are expected to be ready in May 2001.

What does the web services model have to do with components architectures? The web service abstraction has the right distribution characteristics, just like the TP monitor service (transaction) and the EJB component that proceeded it. And SOAP provides an object-oriented access paradigm much like its CORBA, COM and Java RMI predecessors. Web service implementations need a deployment platform as much as any of the others. The only difference is that the web services model is based on a loosely-coupled architecture and, as a result, will be a better fit for inter-business solutions.

4 Conclusions

Component architectures have undergone significant evolution during the last decade. But it is clear that they have become the dominant software development paradigm as we enter the new century. As the Internet grows to encompass pervasive computing devices such as wireless phones, PDAs, GPS signalling devices and more, the demand for new applications will also grow. Component architectures will play a dominant role in the development of these applications.

The transformation that is happening in business today will redefine the business environment of tomorrow. We can expect that more and more of today's business function will be done using the web and that the companies that get there most rapidly will be the ones rewarded with market share, whether they be today's leaders or new entrants. Component architectures will be the critical success factor in making those applications happen.

As the next generation consumers become more comfortable with doing business on the web, they will continue to demand more and better quality services, whether using the PC, cell phone or other more esoteric product currently on the drawing board. All of this software will need to be developed in an ever-decreasing development window. The only technology on the horizon right know that seems to offer the "right stuff" to meet all of these demands is component technology.

Finally, at the risk of being cynical, I observe that even as technology changes, there are certain axioms that remain invariant. Two of the most important:

- coarse-grain collections of programming logic, whether they be called services, transactions, components, or web services are required for effective distribution

- a platform which provides common service as well as the transparent qualities of reliability and scale will always be required to simplify the business programmer's task and future-proof those applications as demand increases.

5 Acknowledgements

A special thanks to Juan Andrade, Bill Cox, Terry Dwyer, and Bob Fleming of BEA Systems for their comments and suggestions that went into the production of this paper.

6 Bibliography

1. [TUX] J. Andrade, M. Carges, T. Dwyer, S. Felts, *The Tuxedo™ System*, Addison Wesley, Reading, MA 1996
2. [WEBL] BEA Systems, *BEA WebLogic Server*, http://edocs.bea.com/wls/docs60/index.html
3. [OOAD] G. Booch, *Object Oriented Analysis and Design with Application*s, *Second Edition*, Benjamin/Cummings Publishing Co.,Redwood City, CA, 1994.
4. [VISI] Borland Corp., *Visibroker*, http://www.borland.com/visibroker/
5. [TPMO] E. Cobb, *TP Monitors and ORBs: A Superior Client/Server Alternative,* Object Magazine, February 1995 pp 57-61
6. [OTTP] E. Cobb, *The Impact of Object Technology on Commercial Transaction Processing*, VLDB Journal Volume 6 Issue 3 (1997) pp 173-190
7. [BXML] ebXML, http://www.ebxml.org
8. [EBPR] ebXML, *Global Manufacturers and Retailers Adopt ebXML*, http://ebxml.org/news/pr_20000911.htm
9. [WEBS] D. Ferguson, R. Kerth, *Websphere as an e-business server*, IBM Systems Journal, Vol. 40, No 1, 2001, pp 25-45
10. [SAGA] H. Garcia-Molina, K. Salem, Sagas, Proceedings SIGMOD International Conference on Management of Data, May 1987 pp 249-260
11. [TPCT] J. Gray, A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers, 1993
12. [CICS] IBM Corporation, *CICS*, http://www-4.ibm.com/software/ts/cics/
13. [ORBX] Iona Technologies, *ORBIX Product Family*, http://www.iona.com/products/orbhome.htm
14. [OOSE] I. Jacobsen, *Object-Oriented Software Engineering—A Use Case Driven Approach*, Addison Wesley Longman, Inc., Wokingham, England, 1992
15. [MNET] Microsoft Corporation, *Microsoft .NET*, http://www.microsoft.com/net/default.asp
16. [MVSB] Microsoft Corporation, Microsoft Visual Basic, http://msdn.microsoft.com/vbasic/
17. [EJB] R. Monson-Haefel, *Enterprise Java Beans™, Second Edition*, O'Reilly & Associates, Sebastopol, CA, 2000

18.[CORB] J. Siegel, *CORBA 3 Fundamentals and Programming, Second Edition,* Wiley Computer Publishing, New York, NY, 2000

19.[OASI] Organization for the Advancement of Structured Information Systems (OASIS), http://www.oasis-open.org

20.[COM] Dale Rogerson, *Inside COM*, Microsoft Press, Redmond WA, 1997

21.[EJB1] Sun Microsystems, *Enterprise JavaBeans™ Specification 1.1*, http://java.sun.com/products/ejb/docs.html

22.[EJB2] Sun Microsystems, *Enterprise JavaBeans Specification 2.0*, http://java.sun.com/products/ejb/docs.html.

23.[J2EE] Sun Microsystems, *Java 2 Enterprise Edition Version 1.3*, http://java.sun.com/j2ee/docs.html

24.[JMS] Sun Microsystems, *Java Messaging Service,* http://java.sun.com/products/jms/docs.html

25.[UDDI] Universal Description, Discovery, and Integration (UDDI), http://www.uddi.org

26.[XML] Worldwide Web Consortium (W3C), *Extensible Markup Language*, http://www.w3.org/XML/

Data Integration Is Harder than You Thought (Extended Abstract)

Maurizio Lenzerini

Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy
lenzerini@dis.uniroma1.it
http://www.dis.uniroma1.it/~lenzerini

1 Introduction

Data integration is a central problem in the design of Cooperative Information Systems. A data integration system combines the data residing at different sources, and provides the user with a unified view of these data, called global schema [6]. The global schema is therefore a reconciled view of the information, which can be queried by the user.

One of the basic decisions in the design of a data integration system is related to the problem of how to specify the relation between the sources and the global schema. There are basically two approaches for this problem. The first approach, called *global-schema-centric* (or simply global-centric), requires that the global schema is expressed in terms of the data sources. More precisely, to every concept of the global schema, a view over the data sources is associated, so that its meaning is specified in terms of the data residing at the sources. The second approach, called *source-centric*, requires the global schema to be specified independently from the sources. The relationships between the global schema and the sources are established by defining every source as a view over the global schema. Thus, in the local-centric approach, we specify the meaning of the sources in terms of the concepts in the global schema. A comparison between the two approaches is reported in [8].

The ultimate goal of a data integration system is to answer queries posed by the user in terms of the global schema. Query processing is considered much easier in the global-centric approach, where in general it is assumed that answering a query basically means unfolding its atoms according to their definitions in terms of the sources [7]. The reason why unfolding does the job is that the global-centric mapping essentially specifies a single database satisfying the global schema, and evaluating the query over this unique database is equivalent to evaluating its unfolding over the sources. On the contrary, in the source-centric approach, we have to deal with the problem that several databases may exist that are coherent with the specification of the data integration system [5].

In this paper we show that, when the global schema contains integrity constraints, even of simple forms, the semantics of the data integration system is best described in terms of a set of databases, rather than a single one, and this

© Springer-Verlag Berlin Heidelberg 2001

implies that, even in the global-centric approach, query processing is intimately connected to the notion of querying *incomplete databases* [9], which is a complex task. In the invited talk, we also illustrate basic techniques for computing the correct answers to a global-centric data integration system whose global schema contains integrity constraints, following [3,4].

2 A Framework for Data Integration

In this section, we describe a formal framework for *data integration systems* (DISs). In what follows, for keeping things simple, we will use a unique semantic domain Δ, composed of a fixed, infinite set of symbols representing real world objects. Thus, all databases in this paper have Δ as domain.

Formally, a DIS $\mathcal{D}$ is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}} \rangle$, where $\mathcal{G}$ is the global schema, $\mathcal{S}$ is the set of source schemas, and $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ is the mapping between $\mathcal{G}$ and the source schemas in $\mathcal{S}$.

We denote with $\mathcal{A}_{\mathcal{G}}$ the alphabet of terms of the *global schema*, and we assume that the global schema $\mathcal{G}$ of a DIS is expressed in the relational model with two kinds of constraints:

- *Key constraints*: given a relation r in the schema, a key constraint over r is expressed in the form $key(r) = \mathbf{X}$, where $\mathbf{X}$ is a set of attributes of r. Such a constraint is satisfied in a database $\mathcal{DB}$ if for each $t_1, t_2 \in r^{\mathcal{DB}}$ with $t_1 \neq t_2$ we have $t_1[\mathbf{X}] \neq t_2[\mathbf{X}]$, where $t[\mathbf{X}]$ is the projection of the tuple t over $\mathbf{X}$.
- *Foreign key constraints*: a foreign key constraint is a statement of the form $r_1[\mathbf{X}] \subseteq r_2[\mathbf{Y}]$, where r_1, r_2 are relations, $\mathbf{X}$ is a sequence of distinct attributes of r_1, and $\mathbf{Y}$ is a sequence formed by the distinct attributes forming the key of r_2. Such a constraint is satisfied in a database $\mathcal{DB}$ if for each tuple t_1 in $r_1^{\mathcal{DB}}$ there exists a tuple t_2 in $r_2^{\mathcal{DB}}$ such that $t_1[\mathbf{X}] = t_2[\mathbf{Y}]$.

We assume that a DIS $\mathcal{D}$ is constituted by n sources, and we denote with $\mathcal{S}$ the set of the n corresponding *source schemas* $\mathcal{S}_1, \ldots, \mathcal{S}_n$. We denote with $\mathcal{A}_{\mathcal{S}_i}$ the alphabet of terms of the source schema $\mathcal{S}_i$ and with $\mathcal{A}_{\mathcal{S}}$ the union of all the $\mathcal{A}_{\mathcal{S}_i}$'s. We assume that the various $\mathcal{A}_{\mathcal{S}_i}$'s are mutually disjoint, and each one is disjoint from the alphabet $\mathcal{A}_{\mathcal{G}}$. Each source schema is expressed in the relational model.

The *mapping* $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ is the heart of the DIS, in that it specifies how the relations in the global schema and in the source schemas map to each other. In this paper, we refer to the so-called global-centric approach. We assume we have a query language $\mathcal{V}_{\mathcal{S}}$ over the alphabet $\mathcal{A}_{\mathcal{S}}$, and the mapping between the global and the source schemas is given by associating to each relation in the global schema a *view*, i.e., a query, over the sources. Thus, the mapping can be seen as a set of pairs of the form $\langle r, V_s \rangle$. The intended meaning of a pair $\langle r, V_s \rangle$ is that the view V_s represents the best way to characterize the tuples of r using the concepts in $\mathcal{S}$.

The triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G},\mathcal{S}} \rangle$ represents the intensional level of the data integration system. In order to specify the *semantics* (extensional level) of a DIS, we start with a database for each of the source schemas, and the crucial point is to specify which are the databases of the global schema. A *source database* $\mathcal{C}$ for $\mathcal{D}$ is a database obtained as a union of n databases $\mathcal{C}_1, \ldots, \mathcal{C}_n$, where each $\mathcal{C}_i$ is a database for the source schema $\mathcal{S}_i$. We call *global database* for $\mathcal{D}$ any database for $\mathcal{G}$. A global database $\mathcal{B}$ for $\mathcal{D}$ is said to be *legal for $\mathcal{D}$ wrt $\mathcal{C}$* if:

– $\mathcal{B}$ is a legal database for $\mathcal{G}$, i.e. it satisfies the constraints in $\mathcal{G}$.
– $\mathcal{B}$ satisfies the mapping $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{C}$, i.e. it satisfies each pair $\langle r, V_s\rangle$ in $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ wrt $\mathcal{C}$. In particular, we say that $\mathcal{B}$ satisfies the pair $\langle r, V_s\rangle$ wrt $\mathcal{C}$, if all the tuples satisfying V_s in $\mathcal{C}$ satisfy r in $\mathcal{B}$:

$$V_s^{\mathcal{C}} \subseteq r^{\mathcal{B}}.$$

Note that the above definition amounts to consider any view V_s as *sound*, which means that the data provided by the sources satisfy the global schema, but are not necessarily complete. Indeed, in this paper, we restrict our attention to sound views only. Other possible choices are described in [1,5].

Given a source database $\mathcal{C}$ for $\mathcal{D}$, the semantics of $\mathcal{D}$ wrt $\mathcal{C}$, denoted $sem(\mathcal{D}, \mathcal{C})$, is defined as follows:

$$sem(\mathcal{D}, \mathcal{C}) = \{\ \mathcal{B} \mid \mathcal{B} \text{ is a legal global database for } \mathcal{D} \text{ wrt } \mathcal{C}\ \}$$

Queries posed to a DIS $\mathcal{D}$ are expressed in terms of a query language $\mathcal{Q}_{\mathcal{G}}$ over the alphabet $\mathcal{A}_{\mathcal{G}}$ and are intended to extract a set of tuples of elements of Δ. Thus, every query has an associated arity, and the semantics of a query q of arity n is defined as follows. Given a source database $\mathcal{C}$ for $\mathcal{D}$, the answer $q^{\mathcal{D},\mathcal{C}}$ of q to $\mathcal{D}$ wrt $\mathcal{C}$ is defined as follows

$$q^{\mathcal{D},\mathcal{C}} = \{(c_1, \ldots, c_n) \mid \text{for all } \mathcal{B} \in sem(\mathcal{D}, \mathcal{C}),\ (c_1, \ldots, c_n) \in q^{\mathcal{B}}\ \}$$

where $q^{\mathcal{DB}}$ denotes the result of evaluating q in the database $\mathcal{DB}$.

Example 1. An example of data integration system is $\mathcal{D}^1 = \langle \mathcal{G}^1, \mathcal{S}^1, \mathcal{M}^1_{\mathcal{G},\mathcal{S}}\rangle$, where $\mathcal{G}^1$ is constituted by the relation symbols $\mathsf{student}(\mathit{Scode}, \mathit{Sname}, \mathit{Scity})$, $\mathsf{university}(\mathit{Ucode}, \mathit{Uname})$, $\mathsf{enrolled}(\mathit{Scode}, \mathit{Ucode})$. and the constraints

$$\begin{aligned} \mathit{key}(\mathsf{student}) &= \{\mathit{Scode}\} \\ \mathit{key}(\mathsf{university}) &= \{\mathit{Ucode}\} \\ \mathit{key}(\mathsf{enrolled}) &= \{\mathit{Scode}, \mathit{Ucode}\} \\ \mathsf{enrolled}[\mathit{Scode}] &\subseteq \mathsf{student}[\mathit{Scode}] \\ \mathsf{enrolled}[\mathit{Ucode}] &\subseteq \mathsf{university}[\mathit{Ucode}] \end{aligned}$$

$\mathcal{S}^1$ is constituted by source s_1 of arity 4, and sources s_2, s_3 of arity 2. Finally, the mapping $\mathcal{M}^1_{\mathcal{G},\mathcal{S}}$ is defined as

$$\begin{aligned} \{ \quad & \langle \mathsf{student}\ ,\ \mathsf{st}(X, Y, Z) \leftarrow \mathsf{s}_1(X, Y, Z, W)\rangle \\ & \langle \mathsf{university}\ ,\ \mathsf{un}(X, Y) \leftarrow \mathsf{s}_2(X, Y)\rangle \\ & \langle \mathsf{enrolled}\ ,\ \mathsf{en}(X, W) \leftarrow \mathsf{s}_3(X, W)\rangle \quad \} \end{aligned}$$

3 Query Answering in the Global-Centric Approach

The ultimate goal of a data integration system is to answer queries posed by the user in terms of the global schema. Generally speaking, query answering over a single database is a well known process. On the contrary, answering a query posed to a system representing a set of databases, is a more complex task. We start this section with an example showing that this is indeed the situation in our setting: the presence of integrity constraints in the global schema leads us to consider multiple databases which are legal for the data integration system.

Example 2. Referring to Example 1, suppose to have a source database $\mathcal{C}$ as shown in Figure 1. Now, due to the integrity constraints in $\mathcal{G}_1$, 16 is the code of some student. Observe, however, that nothing is said by $\mathcal{C}$ about the name and the city of such student. Therefore, we must accept as legal all databases that differ in such attributes of the student with code 16. Note that this is a consequence of the assumption of having sound views, and in order to solve the problem we can think of extending the data contained in the sources in order to satisfy the integrity constraint over the global schema. The fact that, in general, there are several possible ways to carry out such extension implies that there are several legal databases for the data integration systems.

$s_1^{\mathcal{C}}$:

12	*anne*	*florence*	21
15	*bill*	*oslo*	24

$s_2^{\mathcal{C}}$:

12	*AF*
16	*BN*

$s_3^{\mathcal{C}}$:

AF	*bocconi*
BN	*ucla*

Fig. 1. Data at the sources in Example 2

Generally speaking, the presence of integrity constraints may also result in a situation where no legal global database exists. In our setting, this happens, in particular, when the data at the sources retrieved by the queries associated to the relations of the global schema do not satisfy the key constraints in the global schema. Note that this kind of situation is typically addressed by means of suitable data cleaning strategies [2].

As we said in the introduction, it is generally assumed that query answering is an easy task in the global-centric approach. Indeed, the most common technique for query answering in this approach is based on *unfolding*, i.e. substituting to each relation symbol r in the query the corresponding definition in terms of the sources. We now show a simple unfolding strategy is not sufficient for providing all correct answers in the presence of integrity constraints.

Example 3. Referring to Examples 1 and 2, consider the query

$$\mathsf{q}(X) \leftarrow \mathsf{student}(X, Y, Z), \mathsf{enrolled}(X, W).$$

The correct answer to the query is $\{12, 16\}$, because, due to the integrity constraints in $\mathcal{G}_1$, we know that 16 appears in the first attribute of $\mathsf{student}$ in all the databases for $\mathcal{D}$ that are legal wrt $\mathcal{C}$. However, we do not get this information from $\mathsf{s}_1^{\mathcal{C}}$, and, therefore, a simple unfolding strategy retrieves only the answer $\{12\}$ from $\mathcal{C}$, thus proving insufficient for query answering in this framework.

The above example motivates our claim that data integration is harder than one may expect. It shows that, in presence of integrity constraints, even in the global-centric approach we have to face the problem of dealing with incomplete information during query processing. Basic techniques for computing the correct answers to a data integration system whose global schema contains key and foreign key constraints are described in [3,4].

Acknowledgments

Most of the ideas presented in this paper were developed jointly with Diego Calvanese and Giuseppe De Giacomo. I warmly thank both of them. The work presented here was partly supported by MURST Cofin 2000 D2I – From Data to Integration.

References

1. Serge Abiteboul and Oliver Duschka. Complexity of answering queries using materialized views. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 254–265, 1998. 24
2. Mokrane Bouzeghoub and Maurizio Lenzerini. Special issue on data extraction, cleaning, and reconciliation. *Information Systems*, 2001. To appear. 25
3. Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Accessing data integration systems through conceptual schemas. In *Proc. of the 20th Int. Conf. on Conceptual Modeling (ER 2001)*, 2001. To appear. 23, 26
4. Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Data integration under integrity constraints. Submitted for publication, 2001. 23, 26
5. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Answering regular path queries using views. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 389–398, 2000. 22, 24
6. Richard Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'97)*, 1997. 22
7. Alon Y. Levy. Answering queries using views: A survey. Technical report, University of Washinghton, 1999. 22
8. Jeffrey D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT'97)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer-Verlag, 1997. 22
9. Ron van der Meyden. Logical approaches to incomplete information. In Jan Chomicki and Günter Saake, editors, *Logics for Databases and Information Systems*, pages 307–356. Kluwer Academic Publisher, 1998. 23

Implicit Culture for Multi-agent Interaction Support

Enrico Blanzieri[1]*, Paolo Giorgini[2], Paolo Massa[1], and Sabrina Recla[1]

[1] ITC-irst Trento, Italy
{blanzier,massa,recla}@itc.it
[2] Dep. of Mathematics, University of Trento, Italy
pgiorgini@science.unitn.it

Abstract. Implicit Culture is the relation between a set and a group of agents such that the elements of the set behave according to the culture of the group. Earlier work claimed that supporting Implicit Culture phenomena can be useful in both artificial and human agents. In this paper, we recall the concept of Implicit Culture, present an implementation of a System for Implicit Culture Support (SICS) for multi-agent systems, and show how to use it for supporting agent interaction. We also present the application of the SICS to the eCulture Brokering System, a multi-agent system designed to mediate access to cultural information.

1 Introduction

Communication and autonomy are basic capabilities of artificial agents [12], and the agent paradigm suggests that the goals of a Multi-Agent System (MAS) should be reached via the interaction of autonomous communicating agents [11]. In general, interaction is a critical issue. In a brokering system [6,10] for instance the goal of the broker is to facilitate the interaction, which intermediates access to the services of different agents. The overall efficiency of the system is improved through the existance of the broker, because agents can ask the broker for a service without needing to know who are the most suitable agents for performing the given task. In this case, a good interaction policy improves the performances of the system.

Let us consider two sets of agents acting in the same environment with the same goals, but with different skills for performing tasks. For example, in the brokering system we can consider a set of agents that have the capacity to refuse the services provided by unreliable or unsafe agents and another set of agents (or even a single agent) that do not have enough knowledge for doing so. Obviously the latter set of agents performs poorly due to its engagement in unproductive communication. This set of agents can decrease the interaction quality of the MAS even though the other group of agents has good knowledge and skills, i.e. a successful "culture".

* Present: Dep. of Psychology, University of Turin, Italy, blanzier@psych.unito.it

© Springer-Verlag Berlin Heidelberg 2001

The aim of this work is to explore ways of supporting interaction within MAS that takes advantage of the co–presence of sets of agents with different skills. In particular, we focus on the relation between two sets of agents, deriving a strategy and implementing an architecture for improving the interaction. In order to investigate the relation between sets of agents acting consistently, two of the authors introduced the concept of Implicit Culture [2] in a previous work. The definition of Implicit Culture captures the relation between two sets of agents such that the former behaves according to the culture of the latter. Supporting Implicit Culture is an effective way of improving the performances of agents acting in an environment where "better-skilled" agents are also active. Advantages of Implicit Culture support have also been proposed for artificial agents [3], in particular, for controlling the requirements of agent-based systems. Both [2] and [3] have introduced the general architecture of a System for Implicit Culture Support (SICS). No general implemented SICS was given however. In this paper, we present a SICS for multi-agent systems and we show how it is applied to a particular MAS, the eCulture Brokering System.

The paper is organized as follows. In section 2 we describe the concept of Implicit Culture. In section 3 we present a SICS for multi-agent systems, and we show how to use it for supporting agent interactions. Section 4 presents the eCulture Brokering System, a multi-agent system in which the SICS is used. Finally, section 5 gives some conclusions and describes future work.

2 Implicit Culture

When an agent begins to act in an environment, for which it has insufficient knowledge or skills, its behavior is far from optimal. The problems that the new agent must face are made even more complex if other agents are active in the same environment. Other agents would probably have more knowledge and be more skilled. Moreover, they might not be willing to share their knowledge or might not even be able to represent and communicate it. In order to improve its behavior, the new agent should act consistently with the culture of the other agents. Instead, this "new kid in town" is unable to cope with the new environment and with the other agents. Depressingly, a group of agents do have the knowledge and actively exploit it. In the case of humans the phenomenon is sometimes referred to as "cultural shock". Indeed, knowledge about the environment and about the behavior of the agents is part of their culture and that is what the new agent lacks.

The problem of getting the new agent to act consistently with the knowledge and the behaviors of the group could be solved by improving the capabilities of the agent in terms of communication, knowledge and learning. The first solution is just "to ask someone", but in a multi-agent setting, this is *not* a simple solution. In fact, it is necessary to know what to ask (knowledge about the problem), how to ask (a language for expressing the problem), and who to ask (some brokering facility). The second possible solution is to represent the relevant knowledge and to provide it directly to the agent. If the knowledge required is objective and relatively static, this representation can be made by observing the environment and describing it. Building ontologies is a common way to address this problem. Unfortunately, the environment can be partially unknown and intrinsically dynamic. As a third option, it is possible to equip the agent with both observational and learning capabilities allowing it to acquire skills by imitating other agents. The drawback of this techinque is that these capabilities are rather complex and their application requires resources.

When the environment is partially under control, the problem can be tackled in a very different way. Instead of working on the agents capabilities, it is also possible to modify the view that the agent has of the environment and thereby modify its actions. In fact, changing in a proper way the set of possible actions that the agent can perform in the environment can lead the agent to act consistently with the behavior of the group. The group itself can have its behavior optimized on the particular environment. Moreover, neither the new agent nor a member of the group is required to know about the environment and so they share the same culture in an implicit way.

Implicit Culture is the relation between a set and a group of agents such that the elements of the set behave according to the culture of the group. In the following, we informally introduce this notion, while in Appendix A, we restate the formal definition of Implicit Culture presented in [2,3].

We assume that the agents perceive and act in an environment composed of objects and other agents. From this perspective, agents are viewed as objects that are able to perceive, act and know (as a consequence of perception). Actions have as arguments objects, agents or both objects and agents. `Offer(service-1)`, `look_for(broker)` and `request(broker-1,service-1)` are examples of the three different cases, respectively. Before executing an action, an agent is faced with a "scene", which is composed of part of the environment (namely objects and agents) and the set of possible actions the agent can perform on the environment. As a particular case, the agent can also be part of the scene if reflexive actions are possible. For example, an agent `requester-1` could be faced with the environment subset {`broker-1, broker-2, service-1, service-2`} and might be able to perform the actions `request(broker-1,service-1)`, `request (broker-1,service-2)`, `request(broker-2,service-1)`, and `request(broker-2,service-2)`. Hence, an agent executes an action in a given situation (being faced with a given scene at a given time) so we say the agent executes situated actions. The agent `requester-1`,for instance, might execute the action `request(broker-2,service-1)` in a situation where he was facing the scene composed of `broker-1`, `broker-2`, `service-1`, `service-2`.

After each situated action has been executed, the agent faces a new scene. At a given time this new scene depends on the environment and on the previous action executed. If `requester-1` performs `request(broker-2,service-1)`, the `requester-1` will have the scene it faces changed because `broker-2` is now busy and not available to perform any service.

The situated executed action that an agent chooses to execute depends on its private states and, in general, it is not deteministically predictable with the information available externally. Rather, we assume it can be characterized in terms of probability and expectations. As an example, given a `requester` facing a scene in which it can perform `request(the-best-broker,service-1)`, `request(the-worst-broker, service-1)` the expected situated action can be `request(the-best-broker,service-1)`.

Given a group of agents let us suppose that there exists a theory regarding their expected situated actions. If the theory is consistent with the actions executed by the group, it can be considered a cultural constraint for the group. The theory captures the knowledge and skills of the members regarding their environment. For instance:

$$\begin{aligned} &\exists \mathtt{y} \in \mathtt{Group} : \\ &\quad \forall \mathtt{x} \in \mathtt{Group}, \mathtt{s} \in \mathtt{Services}, \mathtt{b} \in \mathtt{Brokers} \\ &\quad \mathtt{request}(\mathtt{x}, \mathtt{y}, \mathtt{s}) \wedge \mathtt{inform}(\mathtt{y}, \mathtt{x}, \mathtt{b}) \rightarrow \mathtt{request}(\mathtt{x}, \mathtt{b}, \mathtt{s}) \end{aligned} \tag{1}$$

expresses that, there exists an agent `y` in the group such that, if a requester `x` requests information of `y` regarding the agents capable of performing a service `s` and `y` replies informing `x` about the broker `b`, then `x` will request `b` to perform `s`. This means that the agent `y` is able to suggest a service provider but also that such a recommendation influences the agents preferences; i.e. `x` decides to request `s` of `b`.

If the set of new agents all perform actions that satisfy the cultural constraints of the group (we call them cultural actions w.r.t. the group), then the problem of the sets suboptimal behavior with respect to that of the group is solved. We describe Implicit Culture as a relation between a set of agents G' and a group of agents G such that the agents of G' perform actions that satisfy a cultural constraint for G. When a set and a group of agents are in an Implicit Culture relation, we have an Implicit Culture phenomenon. As a consequence of an Implicit Culture phenomenon, the actions of a new `requester` of a service will be far more effective if its view of the environment includes only those `brokers` that have previously been requested for the same service.

3 A SICS for Multi-agent Interaction Support

Producing an Implicit Culture Phenomenon is the goal of a System for Implicit Culture Support (SICS) as presented in [2], where a general SICS architecture was introduced. In this section, we present a specific SICS architecture aimed to support multi-agent interaction.

A general SICS (see Figure 1-a) consists of three components: an observer, an inductive module and a composer. The observer stores the executed situated actions of a group of agents G in oder to make them available for the other components. The inductive module uses these actions to produce a cultural constraint theory $\mathbf{\Sigma}$ for G. Finally, the composer, using the theory $\mathbf{\Sigma}$ and the actions, manipulates the scenes faced by a set of agents G' in such a way that their expected situated actions are in fact cultural actions w.r.t G. As a result, the agents of G' execute (on average) cultural actions w.r.t G, and thus the SICS

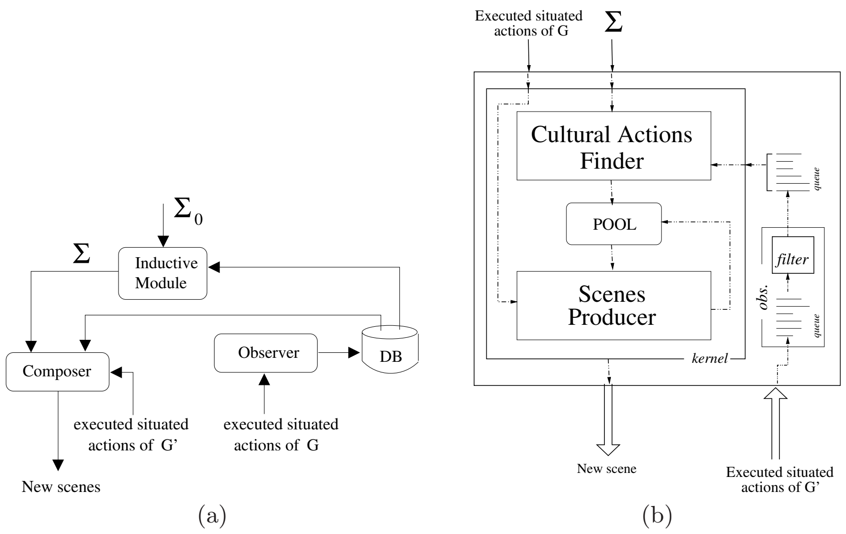


Fig. 1. (a) general architecture for SICS; (b) the composer in detail

produces an Implicit Culture phenomenon. In a multi-agent system, a SICS can be either a general capability of the overall system or a specific capability of a single agent. In the former case, the SICS observes all the agents acting in the system and manipulates the environment. In the latter, the SICS is applied to what the agent is able to observe and change, namely the part of environment and the agents it interacts with by means of actions and communicative acts. The SICS capability, both general and specific, affects the whole system. In the system we present here, we choose to adopt the second option in which a SICS is a capability of a single agent. In order to gain effectiveness we embedded a SICS in a Directory Facilitator (DF), namely an agent that plays a central role in the interactions. In particular, we adopt the idea of DF from FIPA specifications [7] and we extends its capabilities with the SICS.

Following FIPA specifications, a DF is a mandatory agent for an agent platform and provides a yellow pages directory service for other agents on the platform. Every agent that wishes to publicize its services to other agents, requests registration of the service with the DF by providing a description of these services. An agent can query the DF requesting information about the services available and the agents registered for providing such services. A DF does not guarantee the validity of the information provided in response to a request, and it does not do any kind of filtering over the information.

By means of a SICS, the DF can produce the Implicit Culture relation between an agent and the agents that have previously requested information, and provide information that influences the preferences of the agent. As presented in [2], this is a generalization of Collaborative Filtering [8,9], where the similarity among agents is elaborated on the executed actions and not only on ratings. The presence of a SICS decreases the quantity of useless requests, thus improving the overall interaction.

We fully implemented the SICS architecture in Java language using XML for expressing the cultural constraint theory.

Architecture and cultural constraint theory
The SICS embedded in the DF is a particular case of the general one. In this specific case, we do not need any kind of theory induction over the observations, the cultural constraint theory is completely specified and the inductive module is omitted (in Figure 1-a, $\mathbf{\Sigma} \equiv \mathbf{\Sigma}_0$). The cultural constraint theory is very simple. Indeed, we want the DF to recommend agents whose services satisfy the request, namely that the expected situated action of the agent is to accept the recommendation of the DF. In other words, the agent who receives a reccomendation, will request the services of the recommended agents. Referring to the formula (1), with the agent $\mathtt{y}$ specified by the DF, the theory is:

$$\begin{aligned}&\forall \mathtt{x} \in \mathtt{Group}, \mathtt{s} \in \mathtt{Services}, \mathtt{b} \in \mathtt{Brokers}\\ &\quad \mathtt{request}(\mathtt{x}, \mathtt{DF}, \mathtt{s}) \wedge \mathtt{inform}(\mathtt{DF}, \mathtt{x}, \mathtt{b}) \rightarrow \mathtt{request}(\mathtt{x}, \mathtt{b}, \mathtt{s})\end{aligned} \tag{2}$$

In general, our architecture accepts cultural theories expressed by a set of rules of the form:

$$A_1 \wedge \cdots \wedge A_n \rightarrow C_1 \wedge \cdots \wedge C_m$$

in which $A_1 \wedge \cdots \wedge A_n$ is referred to as the antecedent and $C_1 \wedge \cdots \wedge C_m$ as the consequent. The idea is to express that "if in the past the antecedent has happened, then there exists in the future some scenes in which the consequent will happen". Antecedent and consequent are conjunctions of atoms, namely two types of predicates: observations on an agent and conditions on times. For instance, $\mathtt{request(x,y,s,t_1)}$ is a predicate of the first type that says that the agent $\mathtt{x}$ requests of agent $\mathtt{y}$ the service $\mathtt{s}$ at the time $\mathtt{t_1}$; while $\mathtt{less(t_1,t_2)}$ is an example of the second type and it simply states that $\mathtt{t_1<t_2}$. The following rule is used to express the cultural theory (2):

$$\begin{aligned}&\mathtt{request}(\mathtt{x}, \mathtt{DF}, \mathtt{s}, \mathtt{t}_1) \wedge \mathtt{inform}(\mathtt{DF}, \mathtt{x}, \mathtt{y}, \mathtt{t}_2) \wedge \mathtt{less}(\mathtt{t}_1, \mathtt{t}_2) \rightarrow\\ &\quad \mathtt{request}(\mathtt{x}, \mathtt{y}, \mathtt{s}, \mathtt{t}_3) \wedge \mathtt{less}(\mathtt{t}_2, \mathtt{t}_3)\end{aligned} \tag{3}$$

which states that if $\mathtt{x}$ asks the DF for the service $\mathtt{s}$, and the DF replys informing $\mathtt{x}$ that $\mathtt{y}$ can provide such a service, then $\mathtt{x}$ will request of $\mathtt{y}$ the service $\mathtt{s}$.

Observer
The observer observes and stores the actions performed by the agents interacting with the DF. In particular, it stores the requests an agent sends to the DF, the responses that the DF sends to the agent, and the subsequent requests for the service sent by the requesting agent to the recommended service providing agent. The information stored is of the form:

- $\mathtt{request(x,y,s,t)}$, $\mathtt{x}$ sends a request to $\mathtt{y}$ for the service $\mathtt{s}$ at time $\mathtt{t}$;
- $\mathtt{inform(x,y,o,t)}$, $\mathtt{x}$ informs $\mathtt{y}$ about $\mathtt{o}$ at time $\mathtt{t}$.

```
loop
  get the last executed situated action $\alpha$
  for all rule r of $\mathbf{\Sigma}$ do
    for all atom a of r's antecedent ant do
      if match(a,$\alpha$) then
        if find-set(ant,past-actions) then
          join(past-actions,r)
          return r's consequent cons
        end if
      end if
    end for
  end for
  return false
end loop
```

Fig. 2. The algorithm for the CAF submodule

Composer
The goal of the composer is propose a set of scenes to agents of G' such that the expected situated actions of these agents satisfy the cultural constraint theory $\mathbf{\Sigma}$ for the group G. In our case, G' is composed of only one agent (namely the one currently requesting information for the DF), while G is composed of all the agents that have interacted with the DF in the past. The cultural constraint $\mathbf{\Sigma}$ is expressed by the rule (3) and the scenes are composed of the agents recommended by the DF. Figure 1-b shows the composer in detail.

Basically, the composer consists of two main submodules: [1]

- the *Cultural Actions Finder* (CAF), which takes as inputs the theory $\mathbf{\Sigma}$ and the executed situated actions of G', and produces as output the cultural actions w.r.t. G (namely, the actions that satisfy $\mathbf{\Sigma}$);
- the *Scenes Producer* (SP), which takes one of the cultural actions produced by the CAF and, using the executed situated actions of G, produces scenes such the expected situated action is the cultural action.

When an agent `x` asks the DF for a service `s` (i.e. performs the action `request(x, DF,s,t`$_1$`)`) the CAF using (3) finds the cultural action `request(x,y,s,t`$_2$`)`, with `t`$_2$`>t`$_1$, and the SP proposes an agent `y`, of which `x` is expected to request the service `s`.

Cultural Actions Finder
The CAF matches the executed situated actions of G' with the antecedents of the rules of $\mathbf{\Sigma}$. If it finds an action that satisfies the antecedent of a rule, then it takes the consequent of the rule as a cultural action. Figure 2 presents the algorithm for the CAF. For each rule `r` (`ant`$\rightarrow$`cons`), the function *match*(`a`,α)

[1] An additional component of the composer is the *Pool*, which manages the cultural actions given as input from the satisfaction submodule. It stores, updates, and retrieves the cultural actions, and solves possible conflicts among them.

verifies whether the atom `a` of `ant` matches with the executed situated action α; then the function *find-set*(`ant,past-actions`) finds a set `past-actions` of past executed situated actions that matches with the set of atoms of `ant`; and finally, the function *join*(`past-actions,r`) joins the variables of `r` with the situated executed actions in `past-actions`.

Scenes Producer
Given a cultural action α for the agent x (e.g., `request(x,y,s,t`$_2$`)`), the algorithm used in SP consists of three steps:

1. find a set of agents Q that have performed actions similar to α;
2. select a set of agents $Q' \subseteq Q$ similar to x and the set of scenes S in which they have performed the actions;
3. select and propose to x a scene from S.

Steps 1 and 2 are based on two domain dependent similarity and selection functions, and we omit the details. Step 3 selects the scenes in which the cultural action is the expected situated action. To make this selection we first estimate the similarity value between the expected action and the cultural action for each scene $s \in S$, and then we select the scene with the maximum value. The function to be maximized here is the expected value $E(sim(\beta_x, \alpha)|s)$, where β_x is the action performed by the agent x, α is the cultural action, and $s \in S$ is the scene in which β_x is situated. $sim(\beta_x, \alpha)$ is a domain–dependent similarity function and in our case we have chosen:

$$sim(\beta, \alpha) = \begin{cases} 1 & if\ \beta = \alpha \\ 0 & otherwise \end{cases}$$

where $\beta = \alpha$ if the actions are of the same type (namely, `request` or `inform`) and have the same arguments. The following formula is used to estimate E:

$$\hat{E}\left(sim(\beta_x, \alpha)|s\right) = \frac{\sum_{u \in Q'} \hat{E}_1\left(sim(\beta_u, \alpha)|s\right) * w_{x,u}}{\sum_{u \in Q'} w_{x,u}} \tag{4}$$

that is we calculate the weighted average of the similarity value for the expected actions of neighboring scenes. $w_{x,u}$ is the similarity between the agent x and the agent u, while E_1 is estimated as follows:

$$\hat{E}_1\left(sim(\beta_u, \alpha)|s\right) = \frac{\sum_{i=1}^{m} sim(\beta_u^i, \alpha)}{m} \tag{5}$$

that is the average of $sim(\beta_u^i, \alpha)$ over the m actions performed by u in s.

4 The eCulture Brokering System

In this section, we present the eCulture Brokering System, a multi-agent system for brokering cultural information, and to which we have applied the SICS

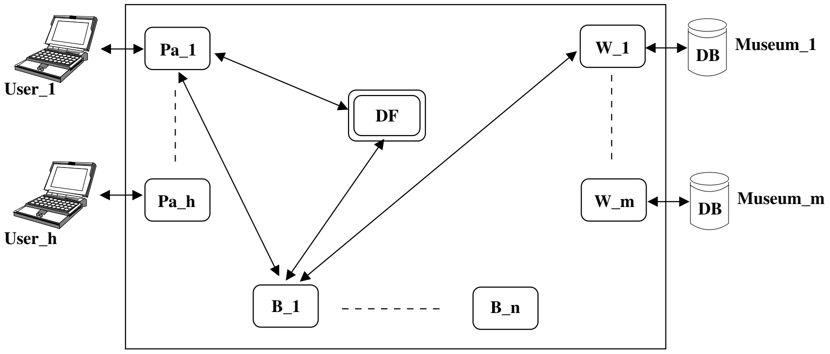


Fig. 3. The eCulture Brokering System

presented above. The multi-agent system has been developed using JACK Intelligent Agents, an agent-oriented development environment built on top of and fully integrated with the Java programming language [4]. The system is the result of a collaboration between ITC-irst and University of Trento. In the same context Busetta et. al [5,1] have developed the basic idea of the implicit support of the SICS architecture presented in [2] for the particular case of messages exchange between agents. They have introduced the innovative overhearing architecture, in which a listener agent observes the messages that two or more parties exchange and a suggester agent composes suggestions and sends them to agents in order to reach the goals fixed *a priori*.

Figure 3 shows a part of the architecture of the eCulture Brokering System (CBS). In particular, it focuses on the agent interaction for which we use the SICS. The platform contains a set of personal agents `Pa_1`,...,`Pa_h`, a `DF` that provides information about a set of brokers `B_1`,...,`B_n`, and a set of wrappers `W_1`,...,`W_m`. A personal agent is created and assigned to each user who accesses the system by means of a web browser. The brokers are specialized in providing information about a specific cultural area (e.g. history, archeology, art, etc...), and they can collect information from different wrappers. Each wrapper is built for a particular database implementation used by a museum. Basically, the databases are of two types: Microsoft Access and Oracle. The complete architecture includes other agents, like for instance the agent resource broker, which provides information about the resources available outside the multi-agent system.

Let us consider a typical interaction. The user can ask the personal agent to search for cultural information about a specific century. The personal agent sends a request to the `DF` for the most appropriate broker that may satisfy the user. The `DF` replys to the personal agent with the name of a broker, and waits for a response message stating whether the personal agent is satisfied with the recommended broker. In case of a negative response, the DF sends the name of another broker to the personal agent and waits again. Having received the name of a broker from the `DF`, the personal agent sends a request to this broker, which

Table 1. Effects of a simple interaction on the state of the DF

Agent	B_1	B_2	B_3	B_4	B_5	B_6
Pa_1	s(XVI)	s(XVI)	u(XVI)	s(VI)	s(IV)	s(VI)
Pa_2	s(XVI)		u(XVI)	s(VI)		
Pa_3	s(VI) u(XVI)	s(XVI)	u(XVI)		u(IV)	u(VI)
Pa_4	s(XVI)		u(XVI)			

in turn asks one or more wrappers to retrieve information from the associated databases.

The DF uses the SICS to decide which broker to suggest to the personal agent. In particular, for each personal agent that sends a request, the DF finds all the agents that have previously performed similar actions (requests and consequent response messages), and then suggests to the personal agent a particular broker with which such similar agents would be satisfied.

Table 1 reports the results of a simple set of interactions between the DF and four personal agents (Pa_1,Pa_2,Pa_3, and Pa_4). In each interaction the personal agent has requested the name of a broker for a specific century (for instance, IV, VI, XVI, ...), and the DF has stored whether its suggestion has satisfied the personal agent (s for satisfied, and u for unsatisfied). Let us suppose that in this situation the personal agent Pa_4 asks the DF for a broker about the VI century. The DF looking at the table finds that agents Pa_1 and Pa_2 are similar to Pa_4, and suggests Pa_4 with the broker that should satisfy Pa_1 and Pa_2, namely with B_4. In case that Pa_4 is not satisfied, the DF will recommend B_6, which has satisfied Pa_1. Of course, this is a very simple case in which it is possible to see which are the similar agents, without any particular elaboration. However, our system is intended to work in complex situations with hundreds of personal agents and, for each of them, tens of requests.

The experiments we have made using the CBS have shown that the SICS can effectively improve the interaction among agents. In particular, it can help new agents (users), that do not know the domain, to interact with the multi-agent system.

5 Conclusion

In this paper, we have presented the concept of Implicit Culture and we have shown how to use a System for Implicit Culture Support for supporting multi-agent interaction. We have presented an implementation of SICS and the eCulture Brokering Systems, a multi-agent system in which the SICS is used.

Implicit Culture Support allows us to improve the interaction among agents without need to equip the agents with additional capabilities. Extending the use of SICS to other agents, such as for instance the agent resource broker (which provides information about the resources available outside the multi-agent system), and implementing the inductive module for inducing cultural constraint theories for different group of agents, are the objectives of our future work.

APPENDIX A: Formal Definition of Implicit Culture

We consider *agents* and *objects* as primitive concepts to which we refer with strings of type *agent_name* and *object_name*, respectively. We define the *set of agents* $\mathcal{P}$ as a set of *agent_name* strings, the *set of objects* $\mathcal{O}$ as a set of *object_name* strings and the *environment* $\mathcal{E}$ as a subset of the union of the set of agents and the set of objects, i.e., $\mathcal{E} \subseteq \mathcal{P} \cup \mathcal{O}$.

Let *action_name* be a type of strings, E be a subset of the environment ($E \subseteq \mathcal{E}$) and s an *action_name*.

Definition 1 (action). *An action α is the pair $\langle s, E\rangle$, where E is the argument of α ($E = arg(\alpha)$).*

Let $\mathcal{A}$ be a set of actions, $A \subseteq \mathcal{A}$ and $B \subseteq \mathcal{E}$.

Definition 2 (scene). *A scene σ is the pair $\langle B, A\rangle$ where, for any $\alpha \in A$, $arg(\alpha) \subseteq B$; α is said to be* possible in σ. *The* scene space $\mathcal{S}_{\mathcal{E},\mathcal{A}}$ *is the set of all scenes.*

Let T be a numerable and totally ordered set with the minimum t_0; $t \in T$ is said to be a *discrete time*. Let $a \in \mathcal{P}$, α an action and σ a scene.

Definition 3 (situation). *A situation at the discrete time t is the triple $\langle a, \sigma, t\rangle$. We say that a* faces *the scene σ at time t.*

Definition 4 (execution). *An* execution *at time t is a triple $\langle a, \alpha, t\rangle$. We say that a* performs *α at time t.*

Definition 5 (situated executed action). *An action α is a* situated executed action *if there exists a situation $\langle a, \sigma, t\rangle$, where a performs α at the time t and α is possible in σ. We say that a performs α in the scene σ at the time t.*

When an agent performs an action in a scene, the environment reacts proposing a new scene to the agent. The relationship between the situated executed action and new scene depends on the charateristics of the environment, and in particular on the laws that describe its dynamics. We suppose that it is possible to describe such relationship by an environment-dependent function defined as follows:

$$F_{\mathcal{E}} : A \times \mathcal{S}_{\mathcal{E},\mathcal{A}} \times T \rightarrow \mathcal{S}_{\mathcal{E},\mathcal{A}} \tag{6}$$

Given a situated executed action α_t performed by an agent a in the scene σ_t at the time t, $F_{\mathcal{E}}$ determines the new scene σ_{t+1} ($= F_{\mathcal{E}}(\alpha_t, \sigma_t, t)$) that will be faced at the time $t + 1$ by the agent a.

While $F_{\mathcal{E}}$ is supposed to be a deterministic function, the action that an agent a performs at time t is a random variable $h_{a,t}$ that assumes values in $\mathcal{A}$.

Let $a \in \mathcal{P}$ and $\langle a, \sigma, t\rangle$ be a situation.

Definition 6 (expected action). *The* expected action *of the agent a is the expected value of the variable $h_{a,t}$, that is $E(h_{a,t})$.*

Definition 7 (expected situated action). *The* expected situated action *of the agent a is the expected value of the variable $h_{a,t}$ conditioned by the situation $\langle a, \sigma, t \rangle$, that is $E(h_{a,t} | \langle a, \sigma, t \rangle)$.*

Definition 8 (party). *A set of agents $G \subseteq \mathcal{P}$ is said to be a* party*.*

Let $\mathcal{L}$ be a language used to describe the environment (agents and objects), actions, scenes, situations, situated executed actions and expected situated actions, and G be a party.

Definition 9 (cultural constraint theory). *The* Cultural Constraint Theory for G *is a theory expressed in the language $\mathcal{L}$ that predicates on the expected situated actions of the members of G.*

Definition 10 (group). *A party G is a* group *if exists a cultural constraint theory $\mathbf{\Sigma}$ for G.*

Definition 11 (cultural action). *Given a group G, an action α is a* Cultural Action w.r.t. *G if there exists an agent $b \in G$ and a situation $\langle b, \sigma, t \rangle$ such that*

$$\{E(h_{b,t} | \langle b, \sigma, t \rangle) = \alpha\}, \mathbf{\Sigma} \not\vdash \perp$$

where $\mathbf{\Sigma}$ is a cultural constraint theory for G.

Definition 12 (implicit culture). Implicit Culture *is a relation $\bowtie$ between two parties G and G' such that G and G' are in relation ($G \bowtie G'$) iff G is a group and the expected situated actions of G' are cultural actions w.r.t G.*

Definition 13 (implicit culture phenomenon). Implicit Culture Phenomenon *is a pair of parties G' and G related by the Implicit Culture.*

We justify the "implicit" term of implicit culture by the fact that its definition makes no reference to the internal states of the agents. In particular, there is no reference to beliefs, desires or intentions and in general to epistemic states or to any knowledge about the cultural constraint theory itself or even to the composition of the two groups. In the general case, the agents do not perform any actions explicitly in order to produce the phenomenon.

References

1. M. Aiello, P. Busetta, A. Donà, and L. Serafini. Ontological overhearing. In *Proc. of 8th International Workshop on Agent Theories, Architectures, and Languages (ATAL 2001), Seattle, USA*, 2001. 35
2. E. Blanzieri and P. Giorgini. From collaborative filtering to implicit culture. In *Proc. of the Workshop on Agents and Reccomender Systems, in Agents2000*, Barcellona (http://www.science.unitn.it/~pgiorgio/ic), 2000. 28, 29, 30, 31, 35

3. E. Blanzieri, P. Giorgini, and F. Giunchiglia. Implicit culture and multi-agent systems. In *Proc. of the Int. Conf. on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, L'Aquila - Italy (http://www.science.unitn.it/∼pgiorgio/ic), 2000. 28, 29
4. P. Busetta, R. Rönnquist, A. Hodgson, and A. Lucas. Jack intelligent agents - components for intelligent agents in java. Technical Report TR9901, AOSs, January. http://www.jackagents.com/pdf/tr9901.pdf. 35
5. P. Busetta, L. Serafini, D. Singh, and F. Zini. Extending multi-agent cooperation by overhearing. In *Proc. the Sixth International Conference on Cooperative Information Systems (CoopIS 2001)*, Trento, Italy, 2001. 35
6. K. Decker, K. Sycara, and M. Williamson. Matchmaking and brokering. In *Proc. of the Second International Conference on Multi-agents Systems (ICMAS-96)*, 1996. 27
7. FIPA. *Foundation for Intelligent Physical Agents.* http://www.fipa.org. 31
8. D. Goldberg, D. Nichols, B. M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992. 31
9. J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997. 31
10. K. Sycara and D. Zeng. Multi-agent integration of information gathering and decision support. In *Proceedings of the 12th European Conference on Artificial Intelligence*, John Wiley & Sons, Ltd., 1996. 27
11. G. Weiss. *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence.* The MIT Press, 1999. 27
12. M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995. 27

Extending Multi-agent Cooperation by Overhearing

Paolo Busetta[1], Luciano Serafini[1], and Dhirendra Singh[2] Floriano Zini[1]

[1] ITC-IRST
via Sommarive 18, 38050 Povo, Trento, Italy
[2] Department of Computer Science, University of Trento
via Inama 5, 38100 Trento, Italy
{busetta,serafini,singh,zini}@itc.it

Abstract. Much cooperation among humans happens following a common pattern: by chance or deliberately, a person overhears a conversation between two or more parties and steps in to help, for instance by suggesting answers to questions, by volunteering to perform actions, by making observations or adding information. We describe an abstract architecture to support a similar pattern in societies of artificial agents. Our architecture involves pairs of so-called *service agents* (or *services*) engaged in some tasks, and unlimited number of *suggestive agents* (or *suggesters*). The latter have an understanding of the work behaviours of the former through a publicly available model, and are able to observe the messages they exchange. Depending on their own objectives, the understanding they have available, and the observed communication, the suggesters try to cooperate with the services, by initiating assisting actions, and by sending suggestions to the services. These in effect may induce a change in services behaviour. Our architecture has been applied in a few industrial and research projects; a simple demonstrator, implemented by means of a BDI toolkit, JACK Intelligent Agents, is discussed in detail.

Keywords: Agent technologies, systems and architectures, Web information systems and services.

1 Introduction

Humans work well in teams, provided the environment is one of communication and collaboration. Often people can produce better results than their normal capabilities permit, by constructively associating with their colleagues. Whenever they are faced with tasks that they cannot manage, or know that can be managed better by other associates, people seek assistance. This observation is not new, and has inspired much research in cooperative agents, for example [6,9,8].

We choose to analyze this association through a slight shift in perspective. While association between agents can readily be achieved by requesting help when required, equal or even improved results can be achieved when associates observe the need for help, and initiate actions or offer suggestions with the

© Springer-Verlag Berlin Heidelberg 2001

aim of improving the plight of their colleague. In fact, these associates may communicate not only when a colleague needs assistance, but also when they feel that they can help improve the productivity of their friend (and hence of their community as a whole).

In this paper, we introduce an abstract architecture for cooperative agents based on a principle that we call *overhearing*. The intuition behind overhearing comes from the modeling of such human interaction as aforementioned, in a collaborative observable environment.

The overhearing architecture has been developed while working on the support of *implicit culture* [3], which offers a very broad conceptual framework for collaboration among agents. Implicit culture is defined as a relation between groups of agents that behave according to a cultural schema and groups that contribute to the production of the same cultural schema. The overhearing architecture described here has a narrower scope than implicit culture, and mostly focuses on agent engineering issues.

One of our goals is supporting a flexible development methodology that prescribes, for its initial phases, the design of only those agents (*services*) required to achieve the basic functionality of a system. The behaviour of the agents, however, can be affected by external observers via *suggestions*, which are special messages carrying information as well as commands. While functionality of the services required by an application are assumed to be immutable, suggesters may be added and removed dynamically without hampering the ability of the system to reach its main objectives.

This has various advantages. Firstly, it is possible to enhance functionality of a running system. As an example, state-of-the-art machine-learning or tunable components can be plugged into the system, as and when they become available, without the need to bring down, rebuild, and then restart the system. Secondly, the output of a system can be enhanced either by suggesting additional related information, or requesting the deletion of outdated or unrelated results. This flexibility comes at the cost of additional sophistication, necessary to perform agent state recognition, to make communication observable, to handle suggestions and to change behaviour accordingly.

This paper is organized as follows. Section 2 presents a pattern of human collaboration that inspired the work presented here. In Section 3 we give some underlying assumptions for our architecture. Section 4 presents the overhearing abstract architecture and identifies what is needed to support it. In particular, it focuses on modelling services' behaviour, on how we realize the overhearing mechanism, and on how suggesters and services can be engineered in order to make and accept suggestions. Section 5 presents a simple system that we developed to demonstrate our architecture. Finally, Section 6 concludes the paper.

2 A Collaboration Pattern

There are several real world examples where our approach to collaboration is practical and appropriate.

Example 1. Consider the scenario where Alice (a for short) asks Bob (b for short) for information on the movies screening at the local cinema. Suppose also that Oscar (o for short) happens to *overhear* this dialogue. In this scenario it is likely that b's reply will be augmented with contributions from o. These contributions (which we call *suggestions*) may either add (or update) information to b's reply or complement it with personal opinion. This is an open communication environment and while a posed the question to b, she may be willing to hear from other friends as well. In general, a may receive an indefinite number of replies to her question, and she is free to deal with them as she pleases.

Example 2. Consider a second scenario with the same actors and the same question posed by a. This time we will assume that b is not up-to-date with the movies screening currently at the cinema. So b decides to ring the reception and find out the details. But as he does, o suggests he better get all details on the actors as well, because a is very likely to ask about them immediately after.

Similar occurrences are common. They are instances of the same pattern, where suggesters are familiar with, and make suggestions to, either or both of the primary conversers. We can relate our real world cases with its actors a, b, and o, to a range of commonly occurring scenarios in the software context. For example:

– a is a search engine building the result page for the user query, b is the enterprise information server specific to, say, fashion, and o is a suggester of such volatile information as fashion shows, exhibitions, and sales.
– a is the operating system, b is the hard disk controller, and o is the cache manager.

Another example, involving more than one suggester, is the following. This case is similar to what we wish to deal with using suggester agents.

Example 3. There are delegations from two separate companies, who have come together for a meeting. Each delegation has a speaker. Other members of the delegation may not speak directly, but may make suggestions to the speaker. In this case, one speaker only talks with the speaker from the other delegation, and makes no attempt to repeat the message for each person in the room. In fact, the speaker makes no assumption on the number of listeners either, since hearing is assumed as long as the listeners are in the room.

3 Underlying Assumptions

For suggestions to be effective, a necessary condition is that a suggester *understands* the behaviour of the suggestee, in order to timely suggest something relevant to its current intentions or future goals. In general, this requires the construction of a behaviour model of the suggestee and the adoption of mental attitude recognition techniques as discussed, for example, in [7,10], based on the observation of its activity.

However, in our approach we assume that *the communicating agents are willing to receive suggestions*, and therefore they make available a *public behavior model* for use by suggesters. These assumptions both reduce the load on the suggesters and improve the quality of their suggestions.

A complementary underlying assumption we make is that *services are aware that their environment is populated by benevolent suggesters* that will not use services' public models to maliciously suggest them. Nevertheless, services are autonomous entities and are so entitled to either accept or refuse suggestions. This can prevent, at least to some extent, misuse of services' public models as well as problems related to timing of delivery, contradictory suggestions from different suggesters and so on.

The assumptions above dramatically simplify the complex problem of mental state recognition, and therefore make it feasible the implementation of real systems, as shown in Section 5.

4 Architecture

Figure 1 summarizes the main components of our architecture. The primary conversers are referred to as *Services* collaborating for their individual needs. The environment could be constituted by a public communication channel between agents, or even by a secure communication mechanism where observation is ex-

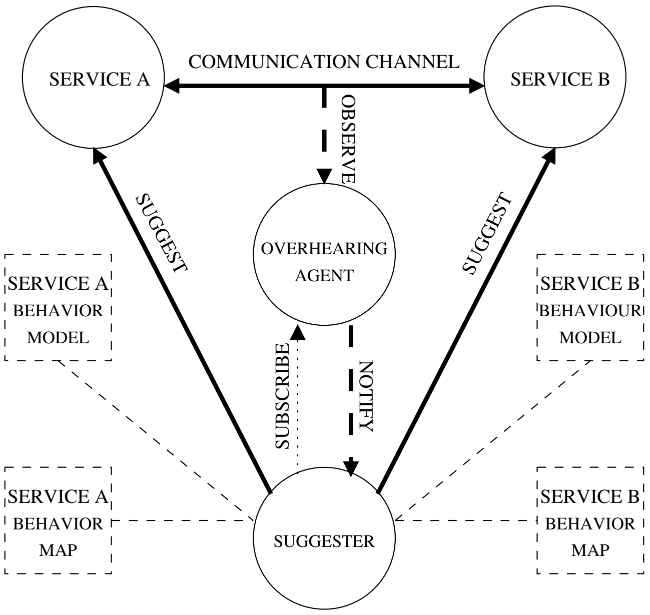


Fig. 1. System Architectural View

plicitly authorized by the communicating services. Other agents (*Suggesters*) are free to participate by listening and, if urge be, to suggest ways in which the goals of the services are better met. The *Overhearing Agent* facilitates conversation overhearing for an indeterminate number of suggesters, and also manages different levels of observation for the various suggesters.

In a normal occurrence, domain specific suggesters would join or leave the system through a subscription service provided by the overhearing agent. Thereon, whenever the overhearing agent hears conversation concerning these domains, this information is forwarded to the respective suggesters. The suggesters are then free to contact the conversing services with suggestions on their operation. In general, the services can expect an indeterminate number of suggestions on their current activity. Most likely, these suggestions will arrive only when suggesters can determine what a service is doing, which is possible whenever the service performs an observable action like conversing with another service.

In order to have its behavior affected, a service needs to make some of its dispositions public. This is realized in terms of a public specification of its behaviors (the "behavior models" in Figure 1). This model can be realized as a finite state machine. The task of the suggesters then, can be formulated as *making suggestions to affect the service agent in a way such that it follows a path in its state space that will satisfy the suggesters own goals.*

In an engineering perspective, the ability to take suggestions enables the design of agents whose behavior can be changed from the outside without touching existing code or restarting a running system.

Once a model is in place, the next issue is: how does a suggester use this knowledge about another agent's behavior? One way is to engineer the model into the suggester. This however makes for very restricted operation in an environment where numerous agents with numerous models may exist. Alternatively, the suggester may be engineered without any information about particular models, but with the ability to understand other models in terms of its own goals (the "behavior maps" in Figure 1). This is clearly very useful for forward-compatibility, as suggesters can be built for models which are still to be invented.

4.1 Modeling Service Agents Behavior

The following considerations are important in our context to define an agent behaviour model.

- The public model need not be an accurate model of an agents behavior. We do not require it to completely match the behavior pattern of the agent. We simply wish to make available *some* representation that can be used by other agents to make suggestions on our agents operation. How this agent interprets this public description of its own behavior is its own decision, and suggesters cannot make any assumptions on this relation. This ensure the decision making autonomy of the agent.

– The communication language between services is strongly related to their individual public models. Indeed, communication between the services is the only means for suggesters to recognize the current state of the services. For this reason, their public model must be expressed with reference to observable communication.
– Since we adopt BDI (Beliefs, Desires, Intentions) as our agent architecture, public models also express agent behavior in terms of public beliefs, intentions, and actions, all or some of which may be affected by suggestions.

In general, the behavior of a BDI agent can be described in terms of transitions between agent mental states. We define the *public model* (model, for short) as a state machine $M = \langle S, T \rangle$, where S is a set of mental states and T is a set of labeled transitions on S, namely $T \subseteq S \times S \times L$, where L is a set of labels. A state $s \in S$ is a triple $\langle B, D, I \rangle$, where B, D, and I denote respectively, *beliefs*, *desires (goals)*, and *intentions* of the agent. Here we do not investigate on the internal structure of the sets B, D and I, we consider them as primitive sets. The set T contains transitions $s \xrightarrow{l} t$ where the label $l \in L$ denotes an *action* that the agent can perform in state s, or an *event* that can happen in the environment. More than one action or event can be undertaken or can happen in a state s. Actually each label $l \in L$ denotes an *action* or *event type* rather than a single action or event. The set of types should be rich enough to give sufficient information about what the service can perform or what can happen in its environment, without going into too many details, to remain at the proper abstraction level.

A state does not necessarily contain *all* the mental attitudes of the service, or the sets of mental attitudes included in a state may not correspond to the "real" ones. In other words, a state contains the representation of the service's mental attitudes it wants to be visible from external observers. Analogously, the set T does not necessarily contain *all* the actions or events the service can perform or perceive, but only the representation of them the service wants to publicize.

The model can contain transitions which corresponds to actions or events that can or cannot be observed by another agent. In our framework communication is the only observable activity of services. The model, however, can refer to other actions that can be undertaken by the service. For instance the model can refer to the action of showing a certain information to the user, or of revising the beliefs, or of retrieving a certain file from a server, etc. These actions do not involve communication with the other agents, and therefore they are not observable by the suggesters. On the other hand suggesters can have some knowledge about these actions, and can give suggestion to the service on the opportunity of taking one of them. The same statement applies to events. The set of labels L is therefore composed of two subset L_o and L_{no} for observable actions/events and non observable actions/events, respectively.

Example 4. A public model for a service acting as an "intelligent" interface to a search engine (described in Section 5) is given by the machine M represented in Figure 2. State s_1 is the initial state, where the service is ready to accept a query (consisting of a keyword to be searched for by the search engine) from

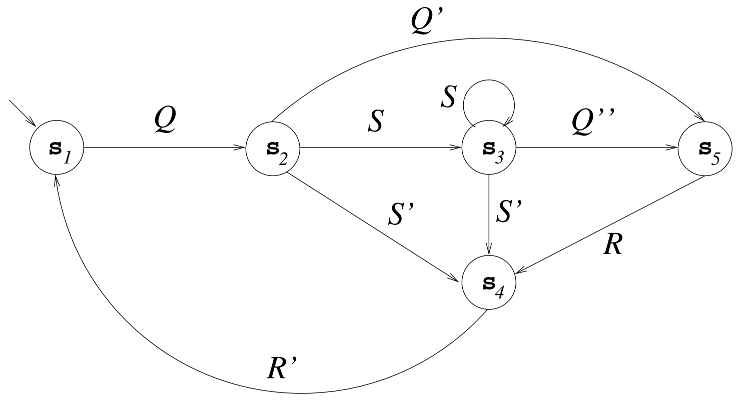


Fig. 2. An example of public model

an user assistant. Receiving the query (observable action Q) leads the service to state s_2 (*ready to submit the query*), from where it can move to three different states:

– if it does not receive any suggestions, it submit the query just received to the search engine (non-observable action Q') and moves to state s_5;
– if it receives a suggestion related to additional keywords to be searched for (observable event S), it moves to state s_3;
– if it is suggested that an answer to the query is already available (observable event S'), it moves to state s_4.

While in state s_3, the service can receive further suggestions on available answers (event S') or additional keywords (event S); eventually, the service submits a query to the search engine which is the composition of Q and S (non-observable action Q''). When in state s_5 (*query submitted*), the service waits for the answer from the search engine (non-observable event R) and moves to state s_4 (*answer available*). From there, the observable action R', consisting of sending a reply to the user assistant with the answer, leads the service back to the initial state s_1.

Suppose that a suggester observes an action Q (a query); it then knows that the service has moved to state s_2 and is ready to submit the user's query to the search engine. Suppose that such a query concerns a subject (say "African Music") which the suggester is expert in; thus, the suggester can provide (additional) information about how to query the search engine for this subject. Technically this means that the suggester sends a message of the form S (a suggestion). In turn, the service can decide to accept or to refuse the suggestion, and thus to submit or not a composed query to the search engine. Note that the refusal case is not shown in Figure 2, since a public model does not need to fully correspond to the actual behaviour of a service.

Another possible reaction of a suggester after observing a query Q is to advise the service on the existence of an alternative (more suitable or faster) information source; for instance, a cache of previous answers. The model above enables a suggester to directly send an answer to the query (suggestion S'), so causing the service to change its state into s_4 and eventually to reply to the user assistant with what has been suggested.

4.2 Overhearing

Consider again the example of the meeting between delegations that we described in Section 2. The question is, how do we take this analogy into computer networks with conversing services and listening suggesters?

A solution, consistent with the human scenario, is that each suggester retrieves the conversation from the underlying network. This solution we will not consider for two main reasons. Firstly, it puts a physical restriction on the presence of the suggesters, because they need to be resident on the same network as the conversing services in order to retrieve messages from it. Secondly, accessing network messages involves network security issues, and does not offer a reliable solution. On the other extreme, each service could keep a record of the suggesters and forward each message of the conversation to each of them as well. This however, is a very expensive exercise as services need to keep track of suggesters who may join or leave the system as they please. Surely, this is not the human way either.

Our solution is an intermediate one. We propose the use of an overhearing agent which would act as an *ear* for all the suggesters in the system. A simple solution then is for services to always send a *duplicate* of the original conversation message to the overhearing agent. The overhearing agent knows about the suggesters within the system and is responsible for sending the message to them.

The advantage is that services can converse without worrying about the number of suggesters in the system. Additionally, services and suggesters are not bound by physical proximity. The overhearer knows about the suggesters, then it may also know about their interests. This means that it is not just a repository of conversation but it sends to the suggesters only messages that are relevant to them. Furthermore, the overhearer may reduce re-analysis on the part of the suggesters, by providing services that perform common analysis *once*, and distributing the results to the suggesters in the system.

The functionality made available by an overhearer to the suggester has a substantial impact on the design of the latter and the types of observations possible on a system. Potential functionality for an overhearer include the following:

Notification of messages. This is the most important functionality required of an overhearer. Suggesters are generally required to specify a criteria (such as a pattern) to be used to select which messages should be forwarded to them. The more sophisticated a selection criteria is, the lesser is the amount of messages being forwarded and the further filtering needed on the service side. On the other hand, the implementation of a sophisticated selection criteria increases the computational cost for the overhearer itself.

Efficient dialogue logging and searching. The dialogue between two service agents could be data intensive. In order to maintain logs of a tractable size, the overhearer should not log every single byte that is exchanged in the dialogue between the two agents. Consider for instance the case where one of the service is a stream provider of movies or music; the overhearer should ignore the data stream itself, and log only meta-data about it.

Dialogue query language. If an overhearer offers a dialogue logging system, it should also provide a query language for the suggesters. This query language cannot be a simple relational query language, since, unlike from a static database, the data collected by the overhearer is a stream, an historical collection. Examples of queries could be *selective queries*, that provide the suggester with some data for each message satisfying a certain property exchanged between a pair of services, or *dynamic queries*, that provide the suggester with all the messages satisfying a property P until a message satisfying the property Q is exchanged.

Simple analysis. This may be required by suggesters as well as a human operator controlling a system. Information to be analyzed may be simply statistical (e.g., number of messages per type and per unit of time), or content-based (e.g., number of messages matching a given pattern). Some of the analysis, as well as part of logs, could be sent periodically to agents performing further elaboration (e.g., collecting databases for collaborative filtering or user profiling).

4.3 Suggesting and Being Suggested

Designing agents able to send or to accept suggestions is a non-trivial task. Indeed, suggesters need to recognize the state of a service, and to decide how to influence the service in order to achieve their own goals. In turn, services that receive suggestions may have to perform reflective reasoning concerning mentalistic aspects (beliefs, goals, intentions), as well as handling interruptions, performing recovery and doing whatever else is necessary when a committed course of actions has to change because of external stimuli. An exhaustive treatment of the issues involved and of the applicable solutions would far exceed the scope of this paper. However, we highlight some of the points that will be addressed by future work.

Provided the public model of a service is viewed as a finite state machine, suggestions can be divided in two general categories, depending on the effects meant to achieve on a service:

– given that the service is in a public state s, suggestions for the choice of a particular action A_i from the list of possible actions $A_1, A_2, \ldots, A_n$ for that public state; and,
– given that the service is in a public state s_j, suggestions for the change in state to s_k, without performing any actions, but through changes in public beliefs, desires and intentions.

Multi-agent model checking [2] and planning as model checking [5] have been adopted as our starting points for the research concerning the reasoning on the current and intended state of a service and on the suggestions to be sent by a suggester.

From the perspective of a service whose internal state corresponds to some public state s, suggestions can be accepted only if it is safe to do so, and must be refused otherwise. Indeed, there are a number of constraints that have to be taken care of; most importantly, consistency in mental attitudes (e.g., avoiding contradictory beliefs or goals, possibly caused by suggestions from different sources) and timing (a suggestion concerning s may arrive when the service has already moved to a different state t).

Particular care is required for the treatment of suggestions that affect a committed course of actions (intention). For instance, a suggestion of adopting an intention i_p (that is, using a plan p on a standard BDI computational model [11]) can arrive too late, after that the agent has already committed to a different intention i_j. Another example is a suggestion that makes some intentions pointless (for instance, intentions of computing or searching the same information being suggested). In these cases, the service should revise its intentions to take advantage of the suggestions. Computationally, the issues with intention revisions are eased by the adoption of *guard conditions* (called *maintenance conditions* in the standard BDI model), which cause a running intention to stop or abort on their failure, that is, when some necessary preconditions are no longer met. However, the effects of actions already performed when an intention is stopped should be properly taken in account, or otherwise reversed or compensated. This issue is, in the general case, hard to deal with, but it is manageable in the case of *anytime algorithms* [12]. An extension of the standard BDI model has also been proposed [4], which exploits the automatic recovery capabilities of distributed nested ACID transactions to handle the same problem for agents operating on databases or other recoverable resources.

5 A Simple Application

We have applied the overhearing architecture to a few research and industrial applications, either to support an implicit culture-based design [3], or as an architectural pattern on its own. For the sake of illustration, we give an overview of a simple demonstrator, developed by means of the JACK Intelligent Agent^{TM} platform by Agent Oriented Software. None of its functionality is particularly innovative, or could not be engineered otherwise; rather, our aim is to show at some level of detail how to put in practice some of the principles described earlier.

The demonstrator acts as an "intelligent" interface to an open source search engine, HTDIG [1]. HTDIG is a world wide web indexing and searching system for an Internet or intranet domain; that is, it works on the contents of one or more web sites.

The interaction between a user and our demonstrator consists of two main phases. In the first phase, the user is shown a list of topics and is asked to pick one or more of them, depending on her interests. In the second phase, the user browses through the list of words extracted by HTDIG during its indexing operations. The user can choose a word from this list; this causes the system to call HTDIG in order to obtain the pointers to the pages containing either the word itself (which is the usual behaviour of HTDIG) or other words related to the chosen one in the context of the topics selected during the first phase, for instance synonims or specializations.

The list of topics shown to the user in the first phase is edited off-line by the system administrator. For each topic, the administrator also edits a simple taxonomy of keywords. These taxonomies are used to enrich the user's requests during the second phase described above. For instance, if the user selects "Software Engineering" and "Literature" as her topics of interest, and then chooses "language" from the list of words, the search is automatically extended to whatever happens to be subsumed by "language" in the Software Engineering and Literature taxonomies, which may include things as diverse as Java, Prolog, English, and Italian.

The demonstrator is implemented as a multi-agent system, including an overhearer able to observe the communication. Four main roles are played in the system: the "user assistant", the "referencer", the "topic suggester" and the "cache suggester". A user assistant is in charge for the interaction with the user, while the actual interfacing with HTDIG is the task of a referencer agent. Once a user has selected a word, her user assistant requests a referencer to search for it; the referencer calls the search facility of HTDIG, then replies to the user assistant with the URL of an HTML page containing the results of the search.

As a matter of fact, the processing of a search request by a referencer is something more than simply calling HTDIG; a simplified model for this activity, sufficient to support the design of – or reasoning by – a suggester agent, is shown in Figure 2. The referencer posts itself a subgoal, `GetReferencePage`, which can be satisfied in three different ways (i.e., by three alternative JACK plans). In the first case, nothing but the word being searched is known; HTDIG is then called for this word. In the second case, the referencer believes that additional keywords are associated to the word; HTDIG is then called to search for the original word and any of the associated keywords. In the third case, the referencer believes that there is already a page with the results of the search; thus, it simply replies with the URL of that page.

Notably, the referencer itself does not contain any logic for associating additional keywords or URLs to a word being searched. However, it is able to accept messages (*suggestions*) concerning those associations. Between receiving a request and submitting the `GetReferencePage` subgoal, the referencer waits for a short time, to allow suggesters to react; if suggestions arrive after the subgoal has been processed, they are ignored.

A topic suggester specializes on the taxonomy of a given topic. It subscribe with the overhearer to be notified of all request messages containing one of the words in its taxonomy, with the exclusion of the leaf nodes. Whenever an assistant sends a request to a referencer, all suggesters for the topics selected by the user react by simply suggesting all the subsumed words.

A cache suggester keeps track of the URLs sent back by referencers, in order to suggest the appropriate URL when a word is searched for the second time by a user (or a user who selected the same topics). To this end, a cache suggester asks to be notified of all requests and replies between user assistants and referencers.

A simple multicasting mechanism has been implemented, which extends the "send" and "reply" primitives of JACK by forwarding a copy of the messages to the overhearer; the latter uses the Java *reflection* package to decompose messages and to match them against subscriptions from suggesters. Subscriptions are expressed as exact matches on contents and attributes of messages, such as sender, receiver, performative, message type and type-specific fields.

This simple demonstrator could easily be enhanced by plugging in additional types of suggesters or enhancing the existing ones, without any change either to the user assistants or to the referencers; consider, for instance, a user profiling module. Building more flexibility – in terms of possible options (e.g. breaking up tasks in subgoals with strategies selected by means of rich beliefs structures) – into the user assistants would potentially enable novel functionality, such as the selection of referencers depending on criteria outside of the control of the system (network or computing loads, collaborative filtering strategies, and so on) that could be tuned or even defined case by case.

6 Conclusions

The overhearing architecture is intended to help and ease the design of flexible, collaborative multi-agent systems, where the main functionality of the system (provided by service agents) are immutable, and additional functions (provided by suggestive agents) may be added and removed dynamically. This is a promising approach when services are willing to accept suggestions from benevolent suggesters in their environment and make a public model of their behavior available.

In future, we expect to deeply investigate some of the major aspects of the architecture: communication infrastructures to support the public communication required for overhearing; modeling, recognition and reasoning on behaviour of service agents; and, computational models for dealing with suggestions.

Acknowledgments

We thank Antonia Donà and Srinath Anantharaju for their contribution to the implementation of the demonstrator, and for reviewing this paper.

References

1. ht://Dig – WWW Search Engine Software, 2001. `http://www.htdig.org/`. 49
2. M. Benerecetti, F. Giunchiglia, and L. Serafini. Model Checking Multiagent Systems. *Journal of Logic and Computation*, 8(3), 1998. 49
3. E. Blanzieri and P. Giorgini. From Collaborating Filtering to Implicit Culture: a General Agent-Based Framework. In *Proc. of the Workshop on Agent-Based Recommender Systems (WARS)*, Barcelona, Spain, June 2000. ACM Press. 41, 49
4. P. Busetta and R. Kotagiri. An architecture for mobile BDI agents. In *Proc. of the 1998 ACM Symposium on Applied Computing (SAC'98)*, Atlanta, Georgia, USA, 1998. ACM Press. 49
5. A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based Generation of Universal Plans in Non-Deterministic Domains. In *Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI-98)*, Madison, Wisconsin, 1998. AAAI-Press. 49
6. J. Doran, S. Franklin, N. Jennings, and T. Norman. On Cooperation in Multi-Agent Systems. *The Knowledge Engineering Review*, 12(3), 1997. 40
7. A. F. Dragoni, P. Giorgini, and L. Serafini. Updating Mental States from Communication. In *Intelligent Agents VII. Agent Theories, Architectures, and Languages - 7th Int. Workshop*, LNAI, Boston, MA, USA, July 2000. Springer-Verlag. 42
8. M. Klusch, editor. *Intelligent Information Systems*. Springer-Verlag, 1999. 40
9. T. Oates, M. Prasad, and V. Lesser. Cooperative Information Gathering: A Distributed Problem Solving Approach. *IEE Proc. on Software Engineering*, 144(1), 1997. 40
10. A. S. Rao. Means-End Plan Recognition: Towards a Theory of Reactive Recognition. In *Proc. of the 4th Int. Conf. on Principles of Knowledge Representation and Reasoning (KRR-94)*, Bonn, Germany, 1994. 42
11. A. S. Rao and M. P. Georgeff. An Abstract Architecture for Rational Agents. In *Proc. of the 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'92)*, San Mateo, CA, 1992. Morgan Kaufmann Publishers. 49
12. S. Zilberstein. Using Anytime Algorithms in Intelligent Systems. *AI Magazine*, 17(3), Fall 1996. 49

Mobile-Agent Based Distributed Web GIS[1]

Jihong Guan[1], Shuigeng Zhou[2], and Aoying Zhou[3]

[1]School of Computer Science, Wuhan University, Wuhan, 430072, China
jhguan@wtusm.edu.cn
[2]Statel Key Lab of Software Engineering, Wuhan University, Wuhan, 430072, China
zhousg@whu.edu.cn
[3]Dept. of Computer Science & Engineering, Fudan University, Shanghai, 200433, China
ayzhou@fudan.edu.cn

Abstract. The widespread use of World Wide Web and the diversity of Geographic Information Systems (GISs) have led to an increasing amount of research on integrating a variety of heterogeneous and autonomous GISs into a cooperative environment to construct a new generation of GISs featuring in open architecture, distributed computing capability, interoperability and extensibility. This paper presents the on-going research project MADGIS, *i.e.*, *M*obile *A*gent based *D*istributed *G*eographic *I*nformation *S*ystem, which aims at integrating distributed Web GIS applications by using mobile agent technologies. The architecture of MADGIS, especially, the structure of client site, server site and mobile agent in MADGIS are described. Key techniques for MADGIS implementation, which include mobile agent building and distributed query processing, are explored. A prototype is established to demonstrate the feasibility of the proposed approach.

1 Introduction

In recent years, the widespread availability of World Wide Web and the diversity of Geographic Information Systems (GISs) have led to an increasing amount of research on integrating a variety of heterogeneous and autonomous GISs into a cooperative environment to construct a new generation of GIS featuring in open architecture, distributed computing capability, interoperability and extensibility [1-5]. From the point of view of GIS interoperability, Y. Bisher *et al* [1] summarize six different levels ranging from network protocols to application semantic; while E. Leclercq *et al* [2] present a different classification which consists of three levels of interoperability: platform level, syntactical level and application level. On the other hand, F.T. Fonseca and M. Egenhofer [3] give a survey of approaches for GIS integration, which range from federated database with schema integration and the use of object orientation to mediators and ontologies. However, as far as system architecture is concerned,

[1] This work has been supported by the Open Researches Fund Program of LIESMARS of China under research contract WKL(00)0302.

© Springer-Verlag Berlin Heidelberg 2001

current solutions for integration of distributed GIS applications are based on either C/S or B/S architectural model [1-5]. But the inherent limitations of these architectures, requiring a proper bandwidth, high quality and stable performance of the network connection, and less supporting of group awareness and high-level cooperation, make them incompetent to fulfill various requirements of dynamic, complicated and distributed computing. Such a situation calls for both new architecture and distributed computing paradigm for establishing distributed GIS under Internet environment.

Mobile agent (MA) is a recently developed computing paradigm that offers a full-featured infrastructure for the development and management of network-efficient applications for accessing information anytime, anywhere, and on both wire-based and wireless devices [6-12]. Mobile agents are programs that can migrate autonomously from one host to another during their execution on behalf of their owners or creators. The agent-based computing can benefit Internet applications on providing asynchronous task execution and more dynamics, supporting flexible and extensible cooperation, reducing communication bandwidth, enhancing real time abilities and higher degree of robustness, enabling off-line processing and disconnected operation. Up to date, several application-oriented mobile-agent systems or projects have been reported in the areas of information retrieval, network management and electronic commence [6-9], *etc*. However, no mobile-agent based system related to spatial data retrieval and manipulation has ever been seen. Considering the advantages of mobile agents over traditional distributed computing technologies, it is worthwhile to introduce mobile agents into distributed GIS building.

In this paper, we present the on-going research project MADGIS (*M*obile *A*gent based *D*istributed *G*eographic *I*nformation *S*ystem), which is a federated database approach based system for integrating distributed Web GISs by using mobile agent technologies. The remainder of the paper is organized as follows. Section2 presents the architecture of MADGIS. Section 3 introduces briefly the implementing techniques of mobile agent in MADGIS. Section 4 discusses the problems of query processing and heterogeneity handling. Section 5 describes a prototype. Section 6 concludes the paper and highlights future research directions.

2 The Architecture of MADGIS

2.1 Overview

The MADGIS system is a cooperative distributed geographic information system, in which mobile agents are responsible for information retrieval from remote data sources. It involves client sites, sever sites, Internet/Intranet connecting these sites, and mobile agents roaming on the Internet/Intranet for retrieving information on behalf of the users. Fig. 1 shows the infrastructure of the MADGIS system.

In Fig. 1, a client site is a client machine, which is used for query submission and results visualization. A server site is also a GIS server, which provides spatial information services for local or remote requests. Precisely, the server site is referred to as *M*obile *A*gent based *D*istributed *GIS* server, or simply MADGIS server. A user submits a query from a client machine to a server *via* web browser. The query is analyzed and optimized by the server, from which one or multiple mobile agents are created and dispatched to finish the query task cooperatively. Each mobile agent along with its sub-tasks travels from one remote server to another to gather the related information. Retrieved information is then taken back to its original site after the mobile agent finish its mission. All returned information is further merged there and then be presented to the user. The MADGIS servers also provide docking facility for mobile agents in case they cannot travel to the destinations promptly due to network problems.

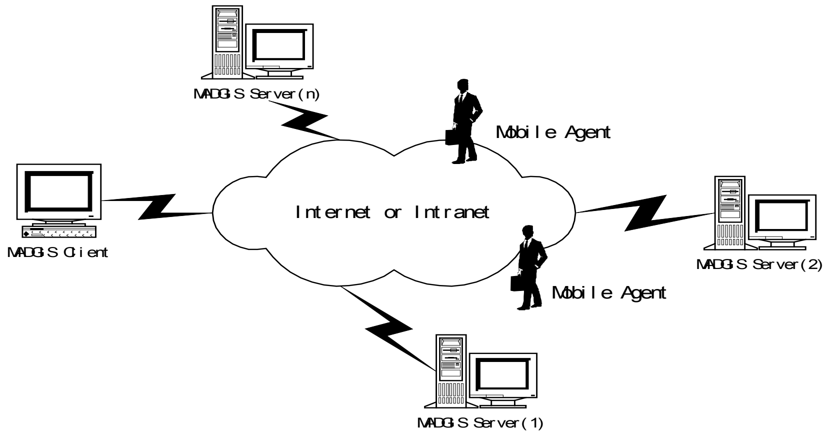


Fig. 1 MADGIS Infrastructure

2.2 The Client Site

A user can access to any GIS site within the MADGIS system *via* a client site. However, the user has to first login a MADGIS server by Web browser. After a successful legal login, the user can start query operations. The accessed MADGIS server returns to the client site a Web page that includes a Graphic User Interface (GUI), from which the user submits queries and gets retrieved information, while the MADGIS server takes charge of query processing and mobile agents manipulation. Typically, a whole query session consists of the following steps:

1) At a client site, the user visits a MADGIS server through Web browser by specifying the server site's URL.
2) The accessed server returns a Web page (usually the server's homepage) that includes a form (or something alike), and waits for the user to authenticate himself/herself by filling the form.
3) The user inputs his/her account and password, and then clicks a button to send the information. The three steps above constitute a normal login process, which happens only in the first query session of a legal user before he/she logouts.
4) If the login is successful, the server returns a Web page that includes a GUI for query submission and results visualization.
5) The user constructs query *via* the GUI and submits it to the server he/she logins.
6) The server receives, analyses, and optimizes the query, and furthermore, creates and dispatches mobile agents to do data fetching. Finally, the retrieved results are returned to the user in HTML format.

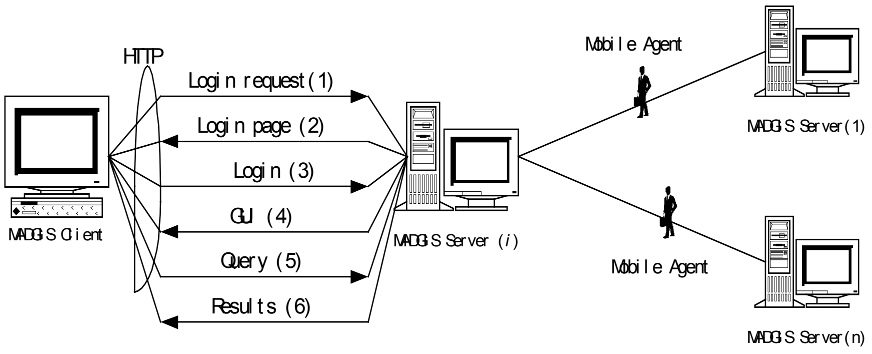


Fig. 2 Client site and its interaction with MADGIS servers and mobile agents

The process described above and the interaction among the client site, server sites and mobile agents are demonstrated in Fig. 2.

Here, the GUI provides transparent access for users to the whole MADGIS system. Typically, GUI includes an area of query edition based on a standard spatial query language specification. Meanwhile, facilities for visually constructing query by filling forms, selecting items from pull-down lists, clicking buttons, and the like, are necessary because most users are not from the GIS domain, so are not familiar with the GIS query language and the query environment.

2.3 The Server Site

A MADGIS server consists of four major components: Web server, Query server, Mobile Agent server (or simply MAserver), and the GIS database. Certainly, there are some other components that are not our main concern here. Fig. 3 illustrates the architecture of a typical MADGIS server.

In Fig. 3, the Web browser at the client site does not directly communicate with the query server at the MADGIS server. Communication between Web server and query server is established through the SQL/Spatial Gateway, *i.e.*, CGI scripts, which is responsible for transferring client request to the query processor in query server. After a query formulated at the client site is submitted, the Web server launches the CGI scripts. Through these scripts, the query processor receives query commands, and begins to interpret query commands, generate query operations and optimize query processing. The goal of optimization is to minimize both computing cost and network traffic volumes incurred by query processing. After optimization, a query plan is generated, which includes a series of sub-queries, their execution sites, and the order in which these sub-queries are executed. If the client query involves only local dataset, then the Spatial Data Engine (SDE) is loaded directly to execute spatial operations. And the query results are sent to MapObjects Internet Map Server (or simply MapObjects IMS), which creates a HTML page including maps in GIF format. The HTML page is finally sent to the client. Here, SDE provides a mechanism to integrate spatial and not-spatial data within GIS Database. MapObjects IMS is used for making dynamic maps, it comprises a Web server extension, and programmable objects that extend MapObjects, and stepwise instructions for building an application that can serve maps to Internet clients. The Global Catalog module in query server provides global metadata and schema that are indispensable to query formulization and optimization.

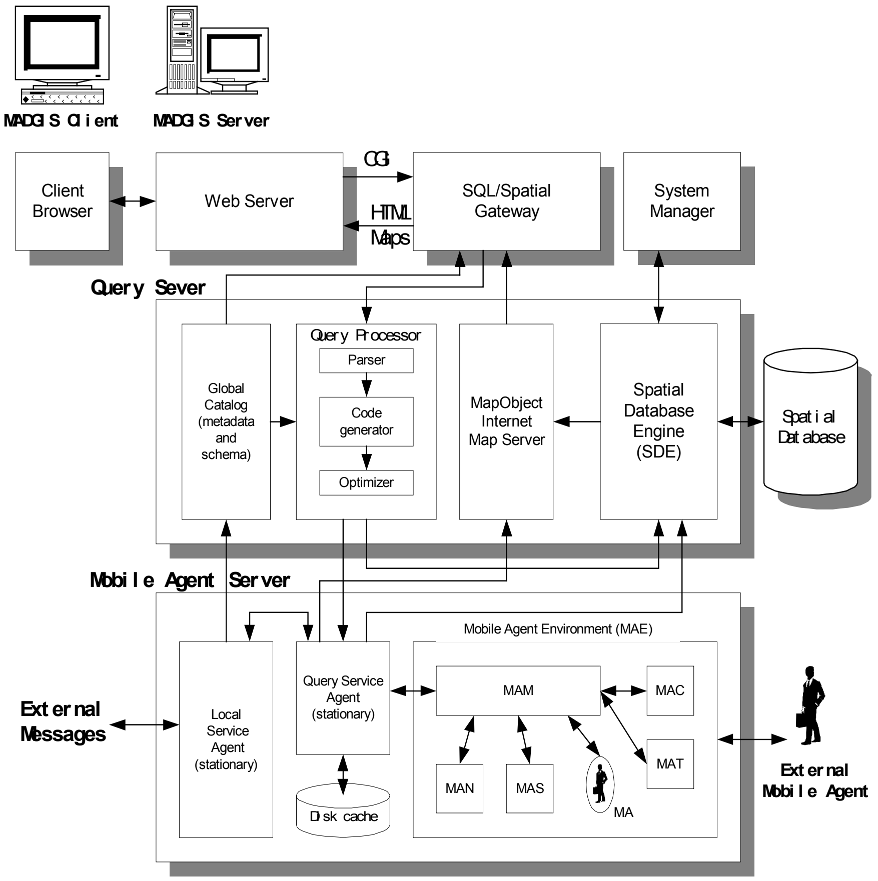


Fig. 3 The MADGIS server site

The MAserver consists of the Local Service Agent (LSA), the Query Service Agent (QSA) and the Mobile Agent Environment (MAE). Here, LSA and QSA are two stationary dedicated agents. LSA is responsible for broadcasting and collecting information about schema and metadata updating occurred in MADGIS servers, and sensing the status of network traffic between the local server and other remote servers. Any changes to schema or metadata in any GIS database within the MADGIS system will lead to the updating of global schema and metadata at each MADGIS server. Network traffic situation will influence the generation of query plan and mobile agents' traveling schedule.

When the client query involves information on remote MADGIS servers, then its query plan is sent to QSA, which is the coordinator of query processing at the local MADGIS server. Its duties include:

(1) Based on the query plan, determining how many mobile agents are needed to gather information at the remote servers and establishing itineraries for all would-be-created mobile agents, then instructing MAE to create and dispatch these mobile agents. In the case of several sub-queries are irrelevant to each other (as far as data is concerned) and network traffic between the local site and remote sites are quite smooth, then several mobile agents are created to do the sub-tasks in parallel, which can lead to better processing efficiency.
(2) Executing the sub-queries that involve only local information.
(3) Combining information from the local site and the remote sites retrieved by mobile agents to construct the final results that are eventually presented to MapObjects IMS for generating HTML page.
(4) Receiving query tasks from mobile agents migrating from remote MADGIS servers, and executing these queries on behalf of the mobile agents to get information from local GIS database.

Above, (1)-(3) is for completing a client query submitted to the local MADGIS server, while (4) is for cooperating with other servers to finish a client query submitted at a remote MADGIS server.

MAE provides an environment for mobile agent creating, executing, dispatching and migrating. Except for the mobile agent, it includes the following functional modules: Mobile Agent Manager (MAM), Mobile Agent Transportation (MAT), Mobile Agent Naming (MAN), Mobile Agent Communication (MAC), and Mobile Agent Security (MAS). Their functions are outlined respectively as follows.

- MAM, the heart of MAE, is responsible for all kinds of management of mobile agents. It mainly provides a comprehensive environment for agent creating and executing, basic functions to make mobile agent code and data migrate precisely to its destination, functions for agent scheduling locally, support for agent's remote management.
- MAT controls the transferring of mobile agents, *i.e.*, sending and receiving mobile agents to and from remote sites. After receiving the instruction of sending a mobile agent from local MAM, MAT authenticates the mobile agent's identification and exchanges cryptographic key with MAE in the destination site, so as to facilitate the destination site to decipher correctly. After packaging, compressing and ciphering the mobile agent, MAT dispatches it. MAT at the destination does inversely, *i.e.*, deciphering, decompressing and un-packaging, and then passes the mobile agent to MAM for further manipulation.
- MAN manages mobile agents' naming service, which provides the mechanism of tracing mobile agents. MAN includes a two-level namespace for mobile agents and allows mobile agents to send messages to each other within this namespace. The first level is the network location of the mobile agent. The second level is a location-unique integer that the server picks for the mobile agent or a location-unique symbolic name that the mobile agent picks for itself.
- MAC serves communication, which provides support for cooperation, events transmission among agents. Here, a message is an arbitrary sequence of bytes with no predefined syntax or semantics except for two types of distinguished messages. An event message provides asynchronous notification of an

important occurrence while a connection message requests, rejects or accepts the establishment of a direct connection. A direct connection is a named message stream between agents and is more convenient and efficient than message passing.

- MAS provides a two-facet security mechanism. On one hand, it is responsible for distinguishing users and authenticating their mobile agents in order to protect server' resources from being illegally accessed or even maliciously attacked. On the other hand, it ensures mobile agents not be tampered by malicious hosts or other agents.

2.4 Mobile Agent

In the context of this paper, a mobile agent is a program that can autonomously migrate to the remote MADGIS servers to gather information on behalf of the server that creates it. Fig. 4 illustrates the structure of a mobile agent and interaction between a mobile agent and MAE of the MADGIS server where the mobile agent stays. Typically, a mobile agent consists of four major parts: *Itinerary*, *Code*, *State* and *Credentials*. *Itinerary* represents mobile agent's tasks, specifically, the query plan in this paper. It lists the remote sites to be visited and the corresponding sub-queries to be carried out at all these sites as well as the optimized traveling path. Mobile agent's *Code* refers to the methods that are used to finish its endowed mission, and *State* means its internal data. Mobile agent's *Credentials* include its identity, owner, home site, code base, and any restrictions imposed on its right. The home site may be used to report status information or to return a malfunctioning agent. The code base is a server responsible for providing any code that the agent needs, which need not exist at its current site. Credentials are protected cryptographically, and are used by MAM for access control.

When the query service agent (QSA) instructs MAM to create a mobile agent, it passes to MAM the query plan that will be used as the itinerary of the would-be-created mobile agent. MAM creates the mobile agent and allows it to migrate. The mobile agent follows a stop-and-query scheme, visiting in turn each MADGIS server site listed in its itinerary. On arriving at a new site, the mobile agent checks its location and determines which sub-query to process at the site. Through the MAM at the visited site, the mobile agent then initiates a query dialog with the QSA at that site and waits for the query results from that site. After getting the query results at the current visited site, the mobile agent then dispatches itself to the next MADGIS server site on its itinerary. When the mobile agent finally returns to the original MADGIS server site where it was created, it reports all query results to the QSA and then disposes of itself.

3 Implementation Issues of Mobile Agent in MADGIS

Mobile agent is a new paradigm for implementation of flexible communication and coordination services in loosely coupled distributed systems that pose special requirements of portability, mobility, collaboration, security and reliability. While implementing mobile agent in MADGIS, we first analyze and compare the advantages and disadvantages of several mobile agent systems, including Aglet [10], Concordia [11], Ajanta [12] and TCL [13], then forge our own technological lines. We outline major design issues of mobile agent in MADGIS as follows.

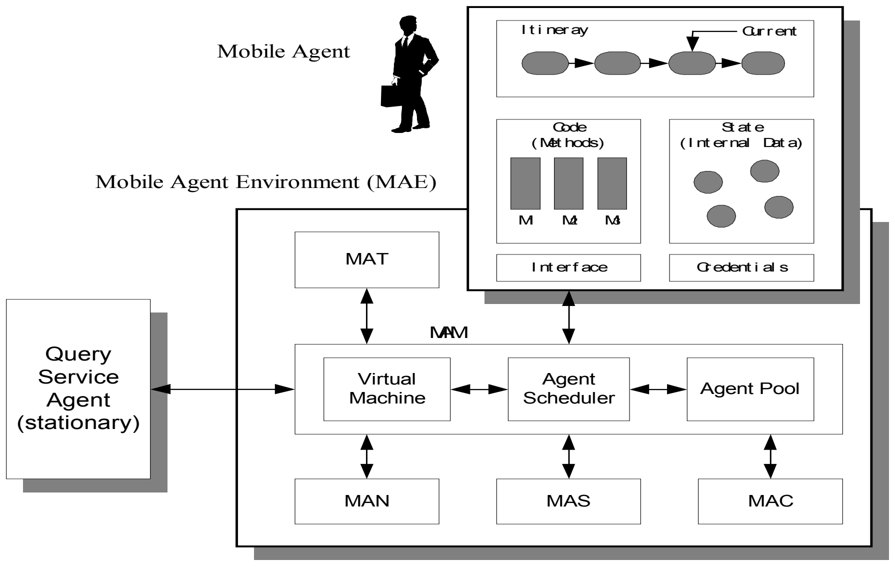


Fig. 4 Mobile agent and MAE

3.1 Basic Mobile Agent Classes

In MADGIS, Java is used to implement mobile agent. We choose Java because it is simple, secure, architecture-neutral, portable, high-performance, multithreaded and dynamic. Such features make it ideal for distributed open applications such as MADGIS. Some basic classes are defined, which forms a hierarchy of mobile-agent related classes. The base class is *BaseAgent*, under which are two sub-classes *SystemAgent* and *UserAgent*. The former represents the abstract class for system to manage mobile agents, and it has several subclasses. For example, *Register* subclass is used to manage the registration and distribution of mobile agents; *QueryAgent* subclass is used to query mobile agent information; *EventManager* subclass is used to monitor various events; *RuleManager* subclass is responsible for defining and managing the trigger rules of different services; and *CollaboratorAgent* subclass takes charge of coordinating the actions of different agents. The latter, *UserAgent*, is used for creating special-purpose, user-defined agents, such as GIS specific agents.

3.2 Mobility

Mobility is one of the essential characteristics of mobile agents. Two different programming styles exist for mobility implementation that reflect different capabilities of the underlying system:

- Non-transparent migration (or weak migration), which assumes that after migration an agent is restarted from the beginning or at a predefined code entry point.
- Transparent migration (or strong migration), which assumes that an agent execution continues on the new target host directly after the instruction that initiated migration.

In our system, transparent migration is implemented. It requires automatic capturing of the entire execution state of an agent, but put much less burden on the agent programmer. We implement transparent migration for mobile agents in the same way as in [14]: A preprocessor which automatically converts Java code written in transparent migration programming style into basic Java, the preprocessor inserts code that saves (and later restores) this information.

3.3 Communication

A set of communication mechanisms are provided in MADGIS:

- The message-based communication mechanism allows agent to send messages to any other agent, whether the agent is local or resides at another server. And the message can be sent asynchronously or synchronously.
- The stream-based communication mechanism allows agents to exchange Java streams that are automatically reconnected when an agent migrates.
- The remote object communication mechanism allows agents to make use of RMI or other communication architectures that offer remote objects. Agents can export such objects or can connect to remote objects that are exported by standard applications.
- The local object communication mechanism allows agents to export and import references to Java objects that are accessible at the local server.

3.4 Security

The Security mechanism of mobile agents deals with the following problems:

- Protecting the hosts without artificially limiting agent access rights.
- Protecting an agent from malicious hosts.
- And protecting groups of hosts that are not under single administrative control.

In MADGIS, security service is provided by the following measures:

(1) Encryption of migrating agents to protect sensitive information within agents.
(2) Digital signatures on migrating agents to authenticate the agent's owner to the new host.
(3) Security policy encoded in MAS to protect system-level resources.
(4) A service proxy (*i.e.*, QSA in this paper) isolates agents from datasets to protect application-level resources.

4 Query Processing

In MADGIS, queries are formulated and processed under the OpenGIS SQL specification [15], which defines a set of standards for the storage and management of spatial data, thus provides a framework for spatial query language design and development.

4.1 Basic Geometry Types and Spatial Functions

In OpenGIS, a set of geometry types is defined as follows. {Geometry, Point, Curve, Surface, GeometryCollection, LineString, Polygon, MultiSurface, MultiCurve, MultiPoint, MultiPolygon, MultiLineString}. Their hierarchical relationship is shown in Fig. 5.

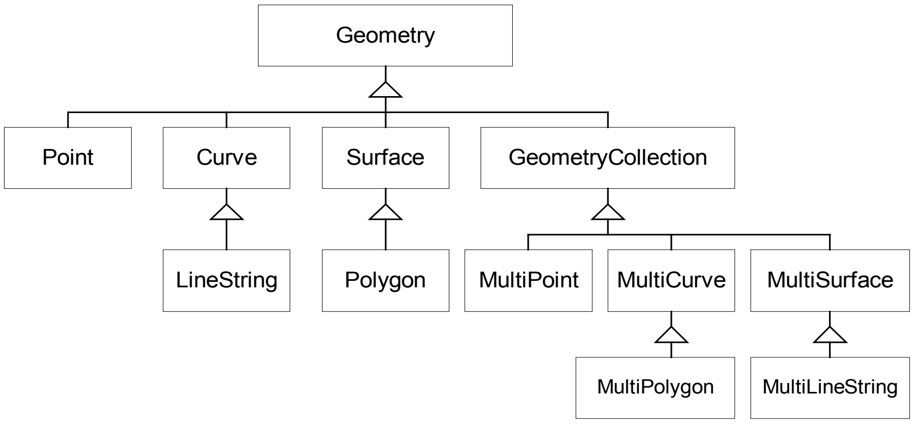


Fig. 5 The hierarchy of geometry types

The root type, named Geometry, has subtypes of Point, Curve, Surface and GeometryCollection. A GeometryCollection is a Geometry that is a collection of possibly heterogeneous geometries. MultiPoint, MultiSurface, MultiCurve are specific subtypes of GeometryCollection used to manage homogenous collections of Points, Curves and Surfaces. Ponit and MultiPoint are zero dimensional geometric types. Curve and MultiCurve together with their subclasses are one—dimensional geometric types. Surface and MultiSurface together with their subclasses are two-dimensional geometric types.

OpenGIS SQL provides an extended Structured Query Language (SQL) to query spatial data. The major extension is the functions for spatial operations, which can roughly be classified into the four groups (some typical functions are listed in Table 1:

(1) Property functions for accessing properties of spatial features.
(2) Relational functions for testing spatial relationships.
(3) Metric functions for measuring distance and direction.
(4) Derivations functions for creating new set of spatial features.

Table 1 Some typical spatial functions

Function group	Function
Property operations	Centroid (Geometry g)
	Area (Geometry g)
	Length (Geometry g)
Relational operations	Pass (Geometry g_1, Geometry g_2)
	Intersect (Geometry g_1, Geometry g_2)
	Within (Geometry g_1, Geometry g_2)
	Contains (Geometry g_1, Geometry g_2)
	On (Geometry g_1, Geometry g_2)
	Adjacent (Geometry g_1, Geometry g_2)
Metric operations	Distance (Geometry g_1, Geometry g_2)
	Direction (Geometry g_1, Geometry g_2)
Derivation operations	Overlap (Geometry g_1, Geometry g_2)
	Buffer (Geometry g, Double d)
	Voronoi (Geometry g)
	Convexhull (Geometry g_1, Geometry g_2)

4.2 Query Representation and Optimization

Generally, a spatial SQL statement is represented in the following form:

Select $A_1, A_2, \ldots, A_m$
From $F_1, F_2, \ldots, F_n$
Where C

Equivalently, it can be represented in relational algebra as

$$\pi_{A_1, A_2, \ldots, A_m}(\sigma_C(F_1 \times F_2 \times \cdots \times F_n)).$$

Therefore, the **SELECT** clause indicates the projection of attributes, the **FROM** clause the Cartesian product of the given relations, and the **WHERE** clause the selection operation in terms of constraint *C*. Constraint *C* can be further written into a conjunctive normal form:

$$C = P_1 \ and \ P_2 \ and \ \cdots \ and \ P_k.$$

Above, P_i ($1 \le i \le k$) is a conjunctive term that may be composed of disjunctive terms. A disjunctive term may be a Boolean spatial function, a "variable-operator-constant" predicate, or a "variable-operator-variable" predicate.

In MADGIS, $F_1, F_2, \ldots, F_n$ may locate at different servers. A query can usually be executed by using several different strategies, and the difference in time requested by different strategies may be as large as several orders of magnitude. Hence in building a practical system, query optimization is inevitable. We developed an improved version of the query optimization algorithm that is first proposed in [5]. Due to space limitation, details of the algorithm are ignored here. Its major steps are as follows.

- Query decomposition: A query is decomposed into a number of sub-queries, each of which will be distributed to a site to be executed.
- Query graph generation: A query graph is constructed to identify data transmissions between sub-queries, in which nodes represent the sub-queries decomposed in the previous step, and edges represent data transmissions between sub-queries.
- Strategies generation: Based on the query graph, alternative strategies are generated by assigning sub-queries to sites. Reduction of network traffic volume is the objective in assigning sub-queries.
- Strategies selection: Considering the costs of data transmission and data processing as well as parallel computing, concrete strategy is selected for final query processing.

4.3 Heterogeneity Handling

In reality, any two GISs may be in different models or/and developed using different software packages, and data in the two GISs may be in different structures and semantic meaning may be expressed in different ways. Such a situation makes handling heterogeneity in data structures and semantics a major barrier in integrating distributed GISs. In MADGIS, we adopt a global schema, a global data format and a global query expression to handle heterogeneity in GIS schemas and geo-spatial data structures as well as semantic conflicts caused by map and attributes names.

The global schema provides definitions of all the features and feature collections accessible to the user. It is based on the OpenGIS specification. So it can be understandable to all data objects in the system. The global data format is based on the Spatial Archive and Interchange Format (SAIF), which uses an object-oriented approach and is possible to model most kinds of spatial data. In the global data format, geo-spatial data are structures as objects and are organized according to a class hierarchy alike to that in Fig.5. Based on the global query expression, a query has three major parts: an input list, an output list and a search condition. Sub-query transferring between different server sites is based on the global query expression.

5 A Simplified Prototype

In order to assess the feasibility of MADGIS, a simplified prototype was developed. For the sake of development time and cost, in the simplified prototype, only part of functions of MADGIS was implemented. For example, query optimization was not implemented because we do not intend to assess query-processing efficiency of MADGIS with this prototype. We use the Aglet Software Development Kit (ASDK) [14] for implementing mobile agents, instead of doing it from scratch according to the technological lines outlined in Section 3. The Query Service Agent (QSA) in Query server accesses to GIS database by calling API functions provided by the GIS development platform on which the GIS server was established. The original GIS server in our prototype was established with a proprietary GIS development platform GeoStar developed by Wuhan University. Java Native Interface (JNI) is used to bridge APIs programmed by C++ at GIS side and QSA implemented by Java at Query server side. Two MADGIS servers are implemented in the prototype, which are termed Sever-A and Server-B. Geographic information of a city is split intentionally into two parts: the spatial data and the non-spatial data, which are stored at Server-A and Server-B respectively. Here, the spatial data indicates geographic information of roads, rivers, buildings and so on, and the non-spatial data includes factual information of hotels, department stores and entertainment sites, *etc.* Usually, a user query involves information from both Server-A and Server-B. A mobile agent is created to retrieve the related information by traveling from Server-A to Server-B, or *vice versa*. Fig. 6 illustrates the GUI and a query example. The GUI is a Chinese interface where menus and buttons are labeled by Chinese. This prototype shows that the architecture of MADGIS is effective to integrating distributed Web GISs.

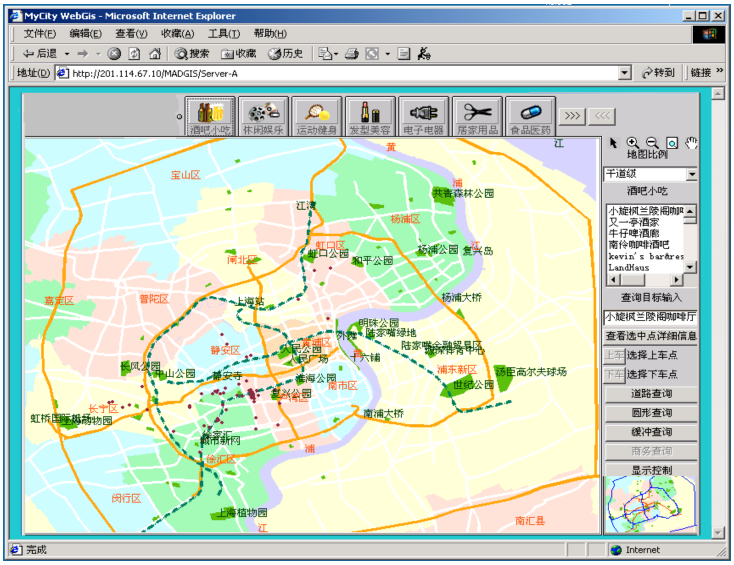


Fig. 6 Query GUI and a query example

6 Conclusion Remarks

Internet has greatly changed the ways of data accessing, sharing and disseminating. The emergence of mobile agent technologies brings new opportunities and challenges to Internet based distributed computing applications. In order to fulfill the requirements of distributed GIS applications under Internet environment with limited communication bandwidth and unstable communication connectivity□we propose a framework for integrating distributed Web-based GISs by using mobile agent technologies, which is termed MADGIS. This paper presents the architecture of MADGIS and concrete technologies for implementation of MADDGIS. A simplified prototype was also established to assess the feasibility of the proposed approach. Further research will be focused on:

– Exploring more efficient architecture of distributed Web-based GIS. A scheme under consideration is to combine Java applets or/and ActiveX technologies into MADGIS.

- Investigating more practical and effective distributed query optimization algorithms.
- Applying the MADGIS framework to real world application systems, including wireless applications

References

1. Y. Bishr. Semantic aspects of interoperable GIS. PhD thesis, ITC, Enschede, NL. 1997.
2. E. Leclercq, D. Benslimane, and K. Yetongnon. Amun: An object-oriented model for cooperative spatial information systems. In IEEE Knowledge and Data Engineering Exchange Workshop, Newport Beach, USA, 1997, pp. 73–80.
3. F. Fonseca, M. Egenhofer. Ontology-driven geographic information systems. In C. Medeiros (ed.) 7th ACM Symposium on Advances in Geographic Information Systems, Kansas City, MO: ACM Press, N.Y., 1999, pp. 14-19.
4. D. Abel, B. Ooi, K. Tan, S. H. Tan. Towards integrated geographical information geoprocessing. Int.Journal of Geographical Information Science, 1998, 12, pp. 353–371.
5. F. Wang. A distributed geographic information system on the Common Object Request Broker Architecture (CORBA). Geoinformatica, 2000, 4 (1), pp.89-115.
6. D. Kotz, R. Gray. Mobile agents and the future of the Internet. ACM Operating Systems Review, 1999, 33(3), pp. 7-13.
7. B. Brewington. Mobile in distributed information retrieval. Intelligent information agent, Matthias Klusch (ed.), Springer-Verlag, Berlin, Germany.1999.
8. K. Kato. An approach to mobile software robots for the WWW. IEEE Transactions on Knowledge and Data Engineering, 1999, 11(4), pp. 526-548.
9. P. Dasgupta. MagNET: Mobile agents for networked electronic trading. IEEE Transactions on Knowledge and Data Engineering, 1999, 11(4), pp. 509-525.
10. Aglet. http://www.tr1.ibm.co.jp/aglets/.
11. D. Wong, N. Paciorek, T. Walsh. Concordia: An infrastructure for collaborating mobile agents. In Mobile Agents: First International Workshop, LNCS, vol. 1219, Springer-Verlag, Berlin, Germany. 1997.
12. N. Karnik, A. Tripathi. Agent server architecture for the Ajanta mobile-agent system. In Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98), 1998.
13. R. Gray. Agent Tcl: a flexible and secure mobile-agent system. PhD thesis, Dartmouth College, 1997.
14. S. Funfrocken. Transparent migration of Java-based mobile agents: capturing and restablishing the state of Java programs. In K. Rothermel and F. Hohl (ed.), Proc. of the Second International Workshop on Mobile Agents (MA’98), LNCS, Springer-Verlag, Stuttgart, Germany, 1998,1477, pp. 26-37.
15. OGC. OpenGIS simple Features Specification for SQL. http://www.opengis.org/thehno/specs.html, 1998.

Local Distributed Agent Matchmaking

Elth Ogston and Stamatis Vassiliadis

Computer Engineering Laboratory, ITS, TU Delft
Delft, The Netherlands
{elth,stamatis}@ce.et.tudelft.nl

Abstract. This paper explores through simulation an abstract model of distributed matchmaking within multi-agent systems. We show that under certain conditions agents can find matches for cooperative tasks without the help of a predefined organization or central facilitator. We achieve this by having each agent search for partners among a small changing set of neighbors. We work with a system where agents look for any one of a number of identical task matches, and where the number of categories of tasks can be as large as 100. Agents dynamically form clusters 10 to 100 agents in size within which agents cooperate by exchanging addresses of non-matching neighbors. We find that control of these clusters cannot be easily distributed, but that distribution in the system as a whole can be maintained by limiting cluster size. We further show that in a dynamic system where tasks end and clusters change matchmaking can continue indefinitely organizing into new sets of clusters, as long as some agents are willing to be flexible and abandon tasks they cannot find matches for. We show that in this case unmatched tasks can have a probability as low as .00005 of being changed per turn.

1 Introduction

In this work we are interested in the global behavior of systems of interacting entities, where interactions take place on a solely local level, and entities make non-deterministic decisions. We consider this setting a good abstract model for large multi-agent systems. [3] Our purpose in studying such systems is to characterize on a conceptual level the behaviors possible in multi-agent systems. We would like our model to include the following properties:

- no agent or other system entity has a global view of the entire system
- agents each know about and communicate with only a small subset of the rest of the system
- agents are built by many different designers
- the system is open and dynamic; agents come, go and change purpose often
- agents are initially placed into the system without considering a global view of the system

We limit the agents in our model to performing exclusively local interactions. By this we mean that agents can only interact within a local neighborhood; that

© Springer-Verlag Berlin Heidelberg 2001

is a small subset of other agents whose addresses are known to that agent. In addition no agent may require any form of knowledge about the entire system, thus we cannot have any central controlling entities. The size of an agent's neighborhood is based upon the resources of the agent, however what we consider important is that as the number of agents in the system grows the size of the neighborhood of any agent in the system remains constant. We create this restriction for a number of reasons. First, it creates a more scalable agent system. Broadcast based algorithms that are required to maintain system wide information are expensive in terms of communication for very large systems. Single node broadcast takes O(r) time, where r is the diameter of the network. In addition to the cost, there are a number of reasons to avoid global information that relate to agent autonomy. Maintaining global information requires cooperation. Thus it requires agents to act on a scope that considers the system as a whole, whereas agents that consider only their individual interests are simpler and much easier to create. Additionally global information is open to sabotage as a malicious agent could corrupt the data that other agents rely on.

A second characteristic of our model is that choices are made nondeterministically. By "nondeterministic" we mean to specify that from the system level view such agent algorithms are forms of randomized algorithms. Agents are modelled as making random choices, though this can include some measure of the agent's environment. Again, this restriction is used to denote several phenomena. First it represents the autonomy of the individual agents; if an agent is truly autonomous we cannot predict how it will behave when observing it from a system level. Second, it represents agent choices that may be made based on incomplete or incorrect data. An agent may have a fixed decision behavior, but may make the wrong choices because it is not possible to for it to know with certainty the data that those choices are based on. Finally, it represents the possible stupidity of agents. Agents may have correct data, yet be limited in their reasoning powers, or even poorly programmed and thus make unexpected choices.

Overall we would like to know what kinds of algorithms a system limited as described above is capable of performing. In this paper we consider a specific problem from this point of view, a form of the matchmaking problem. Matchmaking asks how agents requiring outside services find other agents who are capable of providing those services. We choose the matchmaking procedure for several reasons: it is an essential component of many agent systems, it is well studied, and the majority of algorithms for matchmaking use a central or broadcast component. In a previous paper we defined a local random search procedure for matchmaking and showed that it performed well under a wide range of parameters. [6] However our model had a major drawback in terms of locality; agents were grouped into clusters and their search space was limited to the neighbors of their clusters. Clusters were controlled centrally, and as clusters could grow as large as all the agents in the system, a cluster controller could become a global central controller. In this paper we show that cluster operations cannot be easily distributed, however we find that the size of clusters can be limited. We also extend our model to a dynamic system where tasks complete and agents continue to search for partners for further tasks. We show that such a system deteriorates over time if agents make completely random choices for their new tasks, but can continue indefinitely if agents are flexible and are willing to abandon tasks or consider the partners available when choosing new tasks.

In the remainder of this paper we first discuss related work in sections 2 and our model in section 3. In section 4 we present a summary of our previous results with this model. Section 5 presents new results on decentralizing clusters, limiting cluster size and task completion. Section 6 concludes with some final remarks.

2 Related Work

Matchmaking within multi-agent systems is generally accomplished through a market or by providing middle agents. For one-to-many or many-to-many negotiations market mechanisms are used, as in Contract Net [9]. Here an agent requiring a service broadcasts a request, and providers of that service return bids. Repeated bids and offers can then be used to negotiate fair prices. Markets however can require a large amount of communications as bids and offers are often broadcast to all participants. Moreover, these broadcasts are commonly made though a central market controller that must keep track of all market participants [11]. The second common form of matchmaking within multi-agent systems is through providing a broker or middle agent or agents [2]. There are many different forms of such agents, but they all are based on a set of agents that act as a directory for services available in the system. These directory agents are well know to the other agents, and are queried to find matches. [4] [10]. However this means that middle agents must provide a directory of the entire system. For a large and dynamic system such a directory can be expensive to maintain, and if there are many requests for matches the middle agents can become a bottleneck. On the other hand they provide an efficient solution in terms of numbers of messages. Further, they can guarantee that an agent finds a match if it exists or allow an agent to find the best match system wide. In addition to work on markets and middle agents, there is also a good deal of work on more abstract models of matchmaking. This work is focused on defining the exact cost of creating and maintaining directory servers, and on the levels of distribution possible [1] [5].

In this work we are more curious about how matchmaking can be done on a purely disorganized local level. We explore to what degree matchmaking is still possible among agents that only have knowledge of a few direct neighbors. We find this question interesting as it gives an indication of how well multi-agent systems can organize by themselves, versus how much structure they must have pre-defined for them. It also gives an indication of how well very large unregulated systems might be able to perform from the point of view of an individual agent. One model for such a system is acquaintance networks as described in [7]. This model replaces central directories with broadcast requests made by each agent. We attempt to improve on this by removing the broadcast element, which requires agents to pass on messages for each other, replacing it with a changing neighbors and clustering routine. A related area is the formation of coalitions [8], though again this is mostly focused on forming optimal coalitions and thus also uses system wide broadcast.

3 Model Definition

The model we use in this work in focused on providing an abstract representation of agents attempting to find partners for tasks. Our purpose is to determine under what conditions unorganized agents can find partners, and to minimize any predefined means of cooperating. We would like to determine if such mechanisms are possible given reasonable processing time and memory limitations on individual agents.

Our system consists of a set of agents $A = \{a_1, \ldots, a_n\}$, and a set of task categories $C = \{c_1, \ldots, c_m\}$. Each member of the set of tasks categories, c_i, represents a type of job for which an agent may seek a partner. We further define a pairing among these categories: $f : C \times C \rightarrow [0, 1]$ with

$$f(c_i, c_j) = \begin{cases} 1, \text{ if } (c_i, c_j) \text{ is a matching pair} \\ 0, \text{ otherwise.} \end{cases}$$

We consider the case where each category has only one match and that matches are symmetric. More precisely:

$$\sum_{j=1}^{m} f(c_i, c_j) = 1 \text{ for all } i,$$
$$\sum_{i=1}^{m} f(c_i, c_j) = 1 \text{ for all } j,$$
$$\text{and if } f(c_i, c_j) = 1, \text{ then } f(c_j, c_i) = 1.$$

If $f(c_i, c_j) = 1$ and $i = j$ we have a job that requires the cooperation of two like tasks. More interestingly, we concentrate on the case where $i \neq j$, representing a client-server pair. We show later that these two cases can produce some significantly different behaviors.

In our model, each agent a in A has a set of tasks $T_a = \{t_1, \ldots, t_k\}$, each task belonging to a category in C. T_a can contain more than one task from a category. The goal of the matchmaking problem is for the agents to create links between their tasks and those of other agents, maximizing the number of links between matching client-server pairs as defined above. We shall represent this by defining a graph $G = (V, E)$ where the nodes are all of the agents' tasks and edges are placed between matching tasks. More formally:

$$\begin{aligned} V &= \{(a,t) : a \in A \text{ and } t \in T_a\} \quad \text{and} \\ E &= \{(u,v) \in V \times V : u = (a,t), v = (b,t') \text{ such that if } c \\ &\qquad \text{is the category of task } t \text{ and } c' \text{ is the category} \\ &\qquad \text{of task } t', \text{ then } f(c,c') = 1\}. \end{aligned}$$

Thus G is a graph representing all the possible matches in the system. The matchmaking problem is to find a matching in G, i.e. a set $S \subset E$ such that no two edges in S are adjacent, of maximal size. This represents a system where each task is paired to one other task and the number of matching client-server pairs is maximized. In this work we consider an approximation algorithm for this matchmaking task and instead of searching for a maximum size matching we look for a sufficiently large one.

In our model we initially create random links between tasks. This creates a graph that consists of a random subset of the edges in the complete graph on V above, such that no two edges are adjacent and the degree of each node is 1. In other words each task in each agent is linked to one other task in another randomly chosen agent, giving the agents one neighbor for each task. This represents starting connections formed by some means outside of the system, for instance based on location. Agents then start searching for links that are members of E above, i.e. they look for links that are between two matching tasks. Such connected links represent two agents being able to cooperate. Agents search for connections by permuting which of their unmatched tasks are paired to which of their unmatched neighbors. This is done in turns; during each turn each agent reassigns all of its unmatched tasks and links. When a connection is formed, the agents involved are considered able to cooperate. We assume that agents that can cooperate on one task are likely to be able to work together more closely, for instance they might represent devices from the same manufacturer. Thus, we group such connected agents into a cluster. Clusters then act like compound agents; the unmatched tasks and neighbors of the agents in the cluster are all shuffled together on a turn. Simulations halt after no new connections are formed within a set number of turns.

4 Summary of Previous Results

In a previous paper we reported on the basic behavior of the model described above [6]. Behavior was determined through simulation as the system is too complicated for an exact analysis. Shuffling was done in fixed turns, each agent or cluster shuffling once per turn. During a shuffle links within an agent or cluster were rematched at random with equal probability. In general we found that such a system either quickly formed a single large cluster where almost all links were matches, or remained almost completely unconnected. This behavior varied with the number of categories. An example is shown in Figure 1; for

10 categories almost all matches are found quickly, with 80 categories matches are found quickly, but after an initial slow phase, and with 300 categories most matches are never found. As we increased the number of task categories the probability of this single cluster forming during a trial remained 100 percent until a point at about 90 categories where it drops off suddenly to near 0 percent. We found that this drop off point increases to around 400 categories when each agent is given 4 tasks and 800 categories with 5 tasks per agent. Finally we showed that the system scales well in the number of turns as the number of agents in the system is increased. With 120 categories 2000 agents find clusters in 900 turns while 32000 agents require 1500 turns. For comparison Figures 3, 4 and 5 show data from these original experiments, labelled "original".

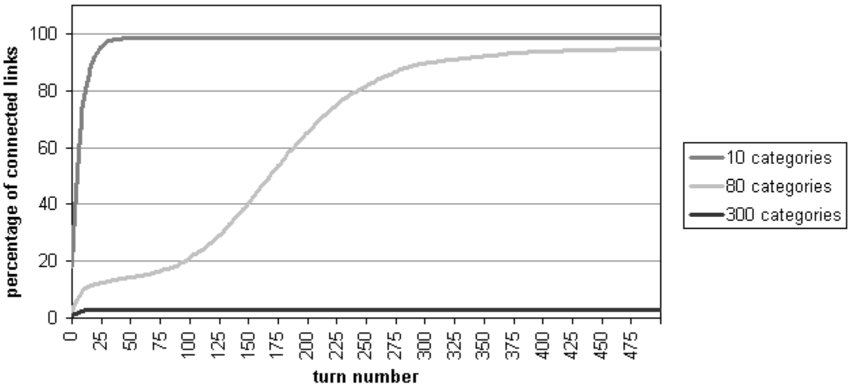


Fig. 1. Sample trials - 2000 agents; 10, 80 and 300 categories

5 Results

Our previous work considered a system where each cluster was controlled centrally. Thus a single entity kept track of the unmatched tasks of a cluster and was responsible for shuffling and reassigning links. As clusters grew in size the number of unmatched tasks in the cluster also increased. Its maximum value approaches 1/3 to 1/2 of all the tasks in the system. Thus the memory need by the central element of a cluster and the time to do a shuffle could become proportional to the number of agents. In the following sections we explore solutions to this problem. We first show in section 5.1 that decentralizing the clusters is not feasible. However, in section 5.2 we find that we can limit the maximum size of clusters and thus keep the system as a whole decentralized. In section 5.3 we study a dynamic system where tasks end and show it always eventually decays. We remedy this in section 5.4 by adding a probability of agents abandoning unmatched tasks. Finally in section 5.5 we briefly describe an unsynchronized version of these experiments. The experiments shown below all are for 2000 agents, each with 3 tasks. All trials are run until the system remains unchanged for 200 turns.

5.1 Decentralizing Clusters

We wish to create a system that remains local, keeping the number of agents each entity in the system knows about and the time any operation takes independent of the number of agents. One way of doing this would be to distribute the control of each cluster. We can change the shuffle operation so that instead of randomly mixing tasks within a cluster, unmatched tasks are given a cyclic ordering and neighbors are simply passed from $task_i$ to $task_{i+1}$ each turn. Experiments using this form of shuffling are labelled "rotate" in Figures 3, 4 and 5 which show that the system still performs well with this form of shuffling. Figure 3 shows, as described above, that our rotating shuffle supports about 20 categories fewer than the original system. Figure 4 shows the percentages of links connected, on average, at the end of the runs in Figure 3. Values for trials that formed a large cluster are shown separately as the "high" data points, and those for trials that did not connect are labelled "low". Here the rotating shuffle performs as well as the original system. Finally Figure 5 shows the number of turns required on average to complete trials that connected. Here the rotate shuffle performs between 14% and 30% better than the original system.

While it is easy to decentralize the rotate operation, we run into problems when forming new matches. When combining two cluster's task sets, A and B, to form a new connection between tasks a_i and b_j a new order can be created by two different methods, as shown in Figure 2. In "Method 1" a_i and b_j are removed leaving the order a_{i-1}, b_{j+1},. . .,b_{j-1}, a_{i+1}. Alternatively in "Method 2" the new order is $a_{i-1}, a_{i+1}, \ldots, b_{i-1}, b_{i+1}$. As Figure 2 shows, which of these methods creates a single cycle depends on whether the connection being formed is between two tasks in the same cluster or tasks in different clusters. Without a cluster center this cannot be easily determined. A broadcast message could be sent through the clusters either asking if the connecting agents are in the same cluster or giving a new cluster ID to agents in one of the clusters. Alternatively the direction of one of the orderings can be reversed, thus creating an identical connection method for each case. However, this takes time proportional to the number of the free tasks in that cluster. Additionally we have further problems if we wish to break a connected link and must know if the cluster it is in remains connected.

5.2 Limiting Cluster Size

Another approach to maintaining locality in our model is to keep clusters centralized, but to limit the size of the clusters. One way to do this would be to connect every matching task pair when a new cluster forms, thus limiting the number of unconnected tasks. This works if matches are made so that category c_i matches itself. However, if matches are between client-server pairs it still leads

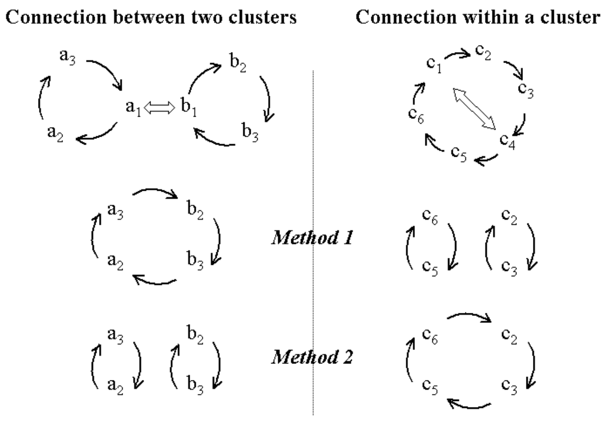


Fig. 2. Two methods of connecting distributed clusters

to large clusters as clusters build up multiple copies of either the client or server in each pair. We found that clusters reached a maximum of between 100 and 300 unconnected tasks when running trials with this method of connection.

In our original experiments we considered only cluster formation, not systems where tasks end thus break up clusters again. Another way of limiting the maximum cluster size would be to adjust the rate at which tasks end so that clusters fall apart before becoming too large. However, the evenness of the rate at which connections are formed makes this adjustment difficult. A high disconnect rates stops most clusters from forming whereas a slightly lower one still allows for large clusters.

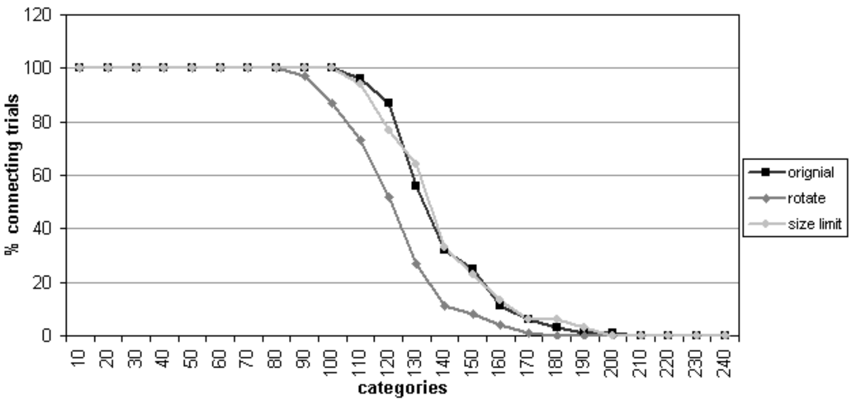


Fig. 3. Percentage of connecting trials: 100 trials per category

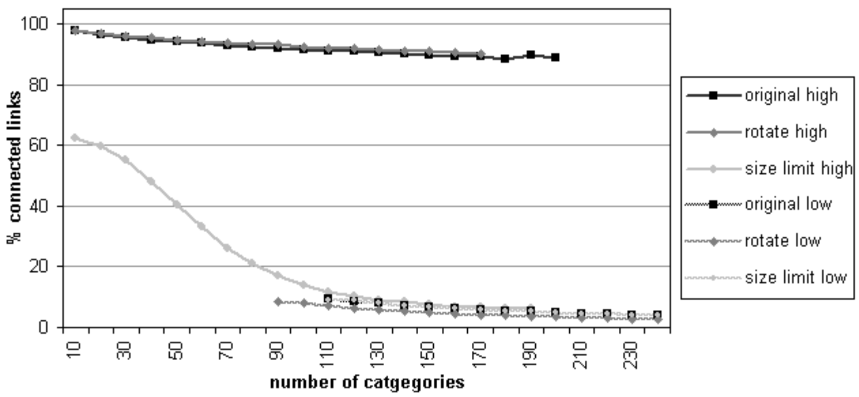


Fig. 4. Average % connected links: 100 trials per category

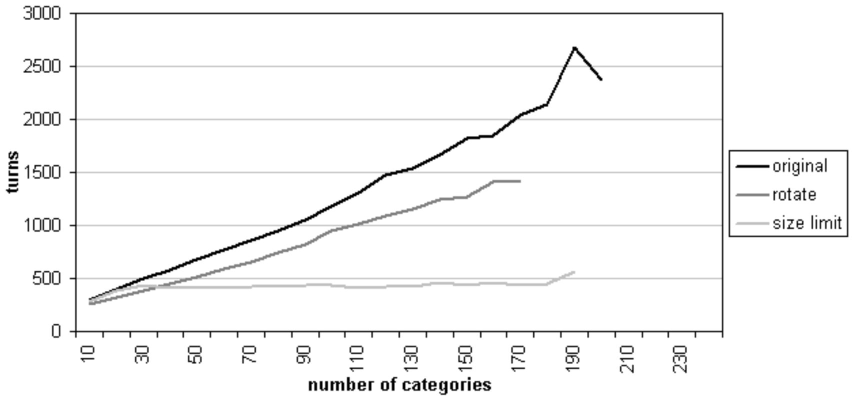


Fig. 5. Average number of turns in a connecting trial vs. number of categories: 100 trials per category

The "size limit" data in Figures 3, 4 and 5 shows results obtained when cluster size is limited by preventing new connections forming once a cluster has reached a given size. The data shown is for clusters limited to 30 agents. Limiting clusters to between 10 and 100 agents provides similar results. The proportions of connected tasks for 100 agent clusters are between one and seven percentage points higher than those shown for clusters of size 30. When cluster size is limited the difference between trials that connect and those that do not is less clear-cut. We separated trials into those that formed a maximum sized cluster and those that did not. Figure 3 shows that the size limited system supports the same number of categories. However Figure 4 shows that as the number of categories increases the percentage of connected links in a trial that forms a "large" cluster decreases towards that of trials that do not connect. We however find in the next section that this limitation does not make as much of a difference in a dynamic system.

5.3 Task Completion

We now extend our model to cover a dynamic system in which tasks are completed and agents then search for partners for new tasks. We add to the system's parameters a probability, P_e, of a connected client-server pair's task ending at a given turn. This represents different jobs that take different amounts of time to complete. When a task between two agents completes, each agent is given a new task of a randomly selected category. The link between these two new tasks is left in place. The cluster center checks if breaking this link breaks up the cluster and if so forms two separate clusters.

Figure 6 shows sample trials with 60 categories, a cluster size limit of 30 and various values of P_e. Things go well for a time, but the number of connections in the system gradually decays and eventually the system falls apart. Figure 7, which shows the category distribution at the start, middle and end of the run with $P_e = .003$, shows why. In this graph client-server pairs are setup so c_0 matches c_1, c_2 matches c_3 etc. At the start of the run the distribution of tasks is pretty much even. At the end however there is a large difference between the numbers of clients and servers in some pairs, thus the number of possible matches in the system is greatly reduced. This happens because matches are made then removed from the system, and eventually a random distribution with a large enough number of members will create a large discrepancy between categories. Since the system needs a minimum number of possible matches, as shown in previous experiments, it will always eventually decay. However, a system where category c_i matches itself avoids this problem and will continue to form connections indefinitely.

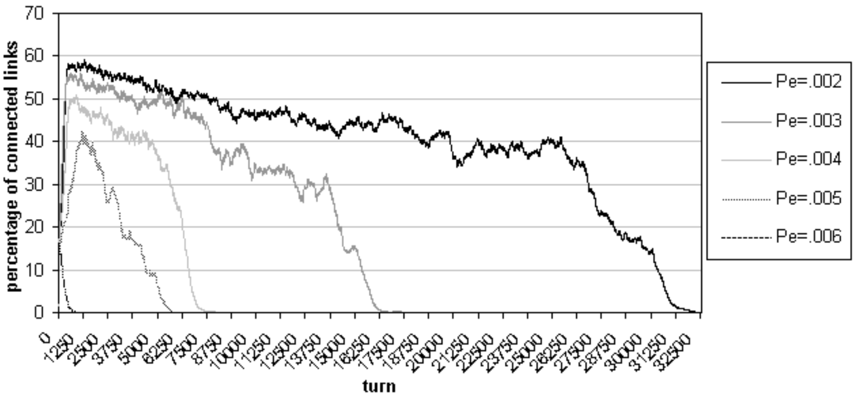


Fig. 6. % of connected links over time when tasks end: 60 categories, size limit 30

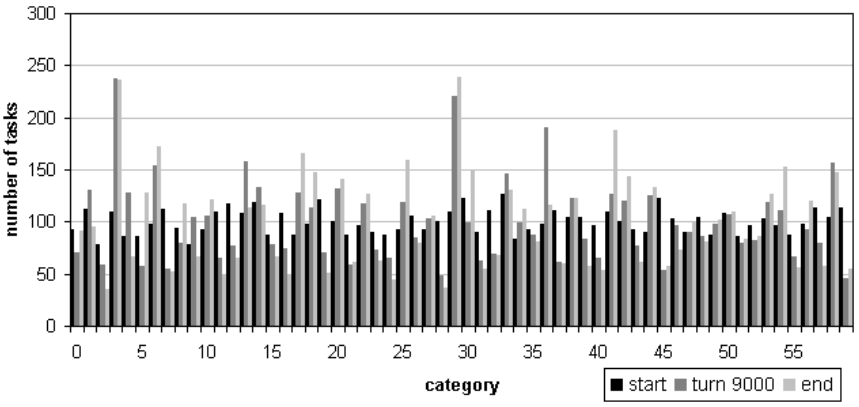


Fig. 7. Category distribution for the P_e=.003 run in Figure 6

5.4 Task Completion with Flexibility

To combat this difference in the number of clients and servers we must add a new element to the system. We create a new parameter P_c, the probability that an unmatched task will change to a new type at each turn. This represents agents giving up on tasks when they cannot find a match for them. Figure 8, which shows sample runs with various P_c values shows that with this mechanism the system no longer decays. Figure 9 shows a longer run for $P_c = .00005$. This run is interesting since it shows a system where clusters form, change, and occasionally completely break apart then form again. Note that in the static case a size limit of 30 produced trials that only formed about 33% of connections, whereas in the dynamic cased they can achieve almost 55% connected links. In a dynamic setting the number of connections at any one time however is not as important. Looking at the tasks completed per agent over time we find that all agents complete comparable numbers of tasks.

5.5 Unsynchronized Turns

Turns in the experiments shown above are synchronized; during each turn each cluster is allowed to move once. Truly decentralized systems do not have such a global clock, thus our experiments assume that all clusters move at the same speed. One way of simulating an unsynchronized system is to move clusters in a randomly chosen order. We repeated the above experiments moving one cluster chosen randomly from all the clusters at a time. For comparison's sake we marked turns as ending after N moves, where N is the number of clusters at the start of the turn. Results for these experiments showed that such an unsynchronized system supported the same number of categories as our original system with less than 1% difference in the number of turns. Sample runs with tasks ending and changing also came out the same as those shown above.

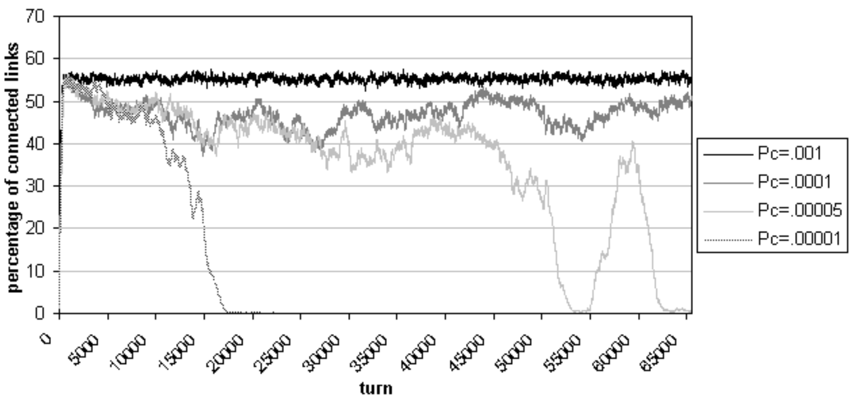


Fig. 8. Percentage of connected links over time when tasks end and unmatched tasks change: 60 categories, size limit 30, P_e=.003

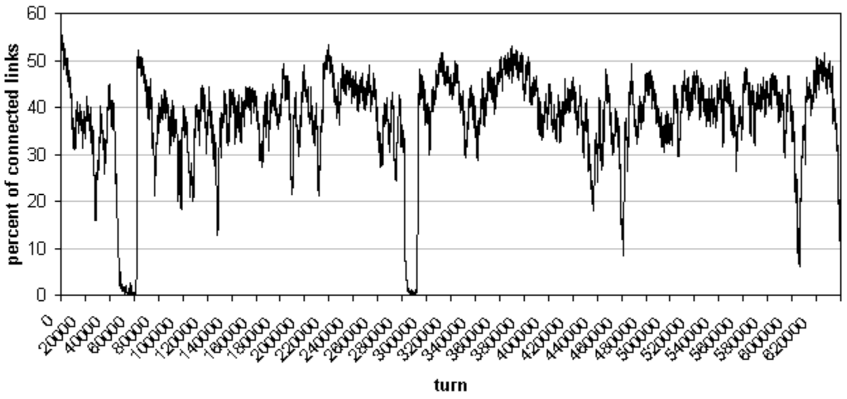


Fig. 9. P_c=.00005, 60 categories, size limit 30, P_e=.003

6 Conclusion

In this paper we showed that a system of un-coordinated agents is able to solve a matchmaking task under certain conditions. We explored a system where agents look for any one of a number of identical matches. Agents searched among a local set of neighbor agents for these matches, and formed cluster partnerships to expand their search space. We showed that such a system could operate if the size of a cluster is limited to as few as 30 agents. We further investigated a dynamic version of this system where client server pairs completed tasks and went on to search for partners for new tasks. We showed that such systems decay for client-server matches because of an eventual uneven client server distribution. We overcame this problem by adding a chance of agents abandoning unmatched tasks. We showed that this probability could be low, .005% per turn per task, and still allow the system to continue indefinitely.

Our abstract model is however still far removed from a real system. Currently the categories of tasks assigned to agents, how long tasks take to complete and the chance of an unmatched task being reassigned are independent random variables. In future work we would like to study systems where these values depend upon the current and past values of other tasks in an agent. This would represent agents carrying out planned series of actions. We would like to know if in such a system all agents are able to complete their programs, or if some will be in positions to perform better than others.

References

1. Awerbuch, B., Peleg, D.: Online Tracking of Mobile Users. Jounal of the ACM, 42(5), (1995) 1021-1058 69
2. Decker, K., Sycara, K., Williamson, M.: Middle-Agents for the Internet. Proceedings of the 15th International Joint Conference on Artificial Intelligence. (1997)578-583 69
3. Ferber, J.: Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, English ed., Addison-Wesley, Edinburgh, (1999) 67
4. Kuokka, D., Harada, L.: Matchmaking for Information Agents. Proceedings of the 14th International Joint Conference on Artificial Intelligence. (1995) 672-678 69
5. Mullender, S., Vitányi, P.: Distributed MatchMaking. Algorithmica, 3, (1988) 367-391 69
6. Ogston, E.,Vassiliadis, S.: Matchmaking Among Minimal Agents Without a Facilitator. Proceedings of the 5th International Conference on Autonomous Agents.(2001)608-615 68, 71
7. Shehory, O,: A Scalable Agent Location Mechanism. Lecture Notes in Artificial Intelligence, Intelligent Agents VI, M. Wooldridge and Y. Lesperance (Eds.). (1999) 162-17 69
8. Shehory, O., Kraus, S.: Methods for Task Allocation via Agent Coalition Formation. Artificial Intelligence, 101(1-2), (1998) 165-200 70
9. Smith, R.: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. IEEE Transactions On Computers, 29(12). (1980) 1104-1113 69
10. Sycara, K., Lu, J., Klusch, M., Widoff, S.: Matchmaking among Heterogeneous Agents on the Internet. Proceedings AAAI Spring Symposium on Intelligent Agents in Cyberspace. (1999) 69
11. Vulkan, N., Jennings, N.: Efficient Mechanisms for the Supply of Services in Multi-Agent Environments. International Journal of Decision Support Systems, 28(1-2) (2000) 5-19 69

Deploying Distributed State Information in Mobile Agent Systems

Ralf-Dieter Schimkat*, Michael Friedrich, and Wolfgang Küchlin

Symbolic Computation Group, University of Tübingen,
D-72076 Tübingen, Germany
{schimkat,friedrich,kuechlin}@informatik.uni-tuebingen.de

Abstract. Agent-based distributed problem solving environments decompose a given problem into a set of sub problems which can be processed in parallel and independently by autonomous and mobile agents at each computation node. Such an autonomous agent primarily makes use of local information which is provided at the respective computation node. This kind of information is characterized by its potential incompleteness and inconsistency with regard to the overall distributed system state which is due to the lack of any centralized coordination facility.
In this paper, we introduce the use of long-term knowledge repositories for autonomous agents without sacrifying the autonomy of the agents and without introducing any central management facility. We compare our approach to an agent-enabled distributed SAT prover which makes only use of local system state information. In that problem solving application a given search tree is distributed dynamically by autonomous mobile agents implemented in pure Java. To demonstrate the profit of using knowledge repositories in general, we integrated our agent system *Okeanos* into our XML-based monitoring system *Specto* and the lightweight, distributed event-based middleware *Mitto*. Our cooperative approach does not conflict with the decentralized parallelization algorithm of the distributed SAT prover. Empirical results show that our approach can contribute to the performance of distributed symbolic computation. In this example, a load balancing subsystem is implemented taking the now available global information about the system state appropriately into account.

1 Introduction

The design and implementation of future distributed systems require a scalable and interoperable software infrastructure in order to provide the necessary computation and connectivity facilities for the participating components. With the advent of the World Wide Web the overall number of participants has increased by an order of magnitude. The concept of software agents will further add to the overall number of participating components. Agents can be considered as rather fine-grained software entities with server capabilities, as opposed to traditional

* supported by T-Systems Document Management Solutions (DMS/IMA)

© Springer-Verlag Berlin Heidelberg 2001

application servers within multi-tier client-server applications. They can much more easily be created and multiplied throughout the system which leads to an enormous increase in the overall number of participants.

In [14] it is argued that cases exist, especially in large-scale computer networks, where interaction among participants residing on remote hosts is rather different in terms of latency and access to application services. Hiding such differences might end up in unexpected performance problems. Therefore the location of system components such as agents in wide-area networks (e.g. the Internet) has to be actively taken into consideration. In the context of mobile agents this includes the capability to reconfigure dynamically, at run-time, the binding between software components (e.g. agents) of the application and their physical location within a computer network, as discussed in [5].

Autonomous agents are characterized by their pro-activeness during the adaptation to changing execution contexts. They carry their own independent interaction plans with them. Such autonomous agents are well suited for distributed applications (e.g. irregular distributed problem solving environments) where the operational environments and task distribution are not completely understood at the time of deployment, but rather emerge as the problem solving task unfolds. The disadvantage of such autonomous systems is the difficulty to avoid situations in which multiple agents pursue similar or equal goals. This might waste computational resources and lead to interdependencies among the participants. Since autonomous agents have only a restricted view on the overall distributed system, it is hard to handle these shortcomings appropriately. An autonomous agent can benefit from distributed state information. Therefore information on the runtime behavior of the distributed system's components have to be collected and made accessible for every autonomous agent. Based on this information, an agent is enabled to broaden its formerly restricted view of the distributed system. This might lead to fewer interdependencies among autonomous agents and end up in optimized interaction plans between them.

In this paper, we introduce so called *long-term XML-based knowledge repositories* to support the efficient deployment of distributed state information for mobile and autonomous agents. These kinds of knowledge repositories are part of a distributed agent-enabled monitoring infrastructure which incorporates a sophisticated, decentralized, and interoperable monitoring facility which is based on XML documents. The paper is organized as follows: Our mobile agent framework called *Okeanos*, the distributed monitoring infrastructure and the distributed satisfiability application are all introduced in Section 2. Then, in Section 3, the empirical results based on the satisfiability checker are presented followed by a general discussion of the deployment of distributed state information for mobile agents. Finally, the paper concludes and gives a short overview of future work.

2 Architectural Overview

In this section the mobile agent framework *Okeanos*, the agent-enabled distributed monitoring infrastructure and finally a distributed satisfiability application are presented. All software systems were developed in Java in our group and are presented in detail in [10,11,12].

2.1 Mobile Agents in *Okeanos*

Okeanos [10] is an agent-based middleware framework for processing and distributing complex tasks. A framework forms an abstract set of classes and interfaces among them, which, taken together, set up a generic architecture for a family of related applications [7]. In general, the *Okeanos* framework provides an infrastructure for mobile agents which may access services of any kind. Each *Okeanos*-agent communicates by passing messages in KQML [3] locally.

The key design goals in *Okeanos* are the support of highly distributed applications within computer networks and high-level interfaces for the integration of various services, for example services for distributed symbolic computations. The services might be implemented in a wide range of programming languages.

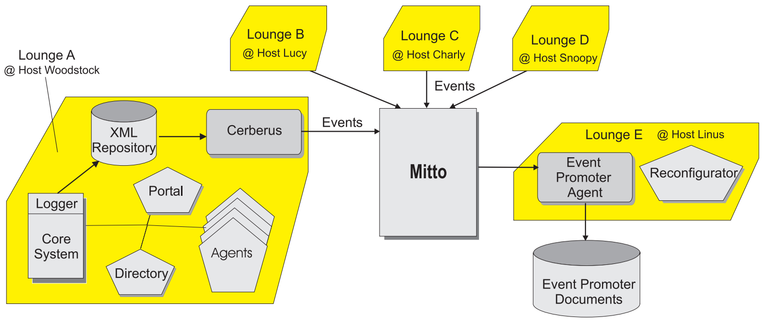


Fig. 1. Agent Monitoring Architecture

Agents In *Okeanos* there are two kinds of agents: stationary and mobile agents. A stationary agent is located at one agent execution environment which it can not leave. Its primary task is to implement and to provide services for other agents. In contrast to stationary agents, mobile agents are characterized by their ability to move among connected nodes.

While the whole agent system, consisting of the execution environment called *Lounge* (see Figure 1), and the agent framework is implemented in pure Java, legacy-systems or -libraries written in other programming languages can easily be made available. These external services are wrapped by dedicated service agents so that they are indistinguishable from other agents.

A rather difficult task to implement is the control logic of mobile agents. Not only the service function has to be composed but also the coordination issues regarding the location and the environmental context have to be taken care of. Therefore, we extended the base agent class and incorporated an rule-based expert system shell named JESS [4] to support this complex job. This leads to a rule-based implementation of functionality which eases the capturing of different states and error handling. Therefore, agents can be built by a set of rules and facts which are carried along as the agent moves around. During its lifetime, the agent gathers new facts and if a matching rule is found, actions based on this knowledge are taken.

Lounge As mobile agents are nothing more than a bunch of bytes to the operating system, most agent systems need execution environments for the deployed agents to be run in. This environment is called the *Okeanos*-Lounge. In addition to processing time and memory it also offers some vital services to the agents:

The *directory service* acts as yellow pages for agent services. It is divided into a global and a local part. Every agent can register at this directory. The lounge and each service form a tuple which is registered in the global part at the directory service. In order to contact a service which is not available locally at the current lounge, an agent has to migrate to a remote lounge offering this particular service. Finally, this agent asks for the stationary service agent which registered that requested service and starts off the service request. That is the primary approach of mobile and autonomous agents in *Okeanos* to locate services within a distributed environment.

The other vital lounge-service, the *portal service*, is in charge of agent migration and persistence. During the process of migration an agent copy is made persistent to external storage. In case of any malfunction this particular agent can be restored using that disk-image. The agents themselves are transferred using RMI which is the single point of remote communication between lounges.

2.2 Distributed Monitoring Infrastructure

In this section, the distributed monitoring architecture called *Specto* is described. Whereas in [11] the general monitoring model of *Specto* is discussed, the focus in this paper is on the integration and "agentification" of *Specto* into *Okeanos* and the necessary architectural enhancements.

Basically, the monitoring infrastructure senses an agent and the agent execution environment (lounge), detects important states of an agent and events, and disseminates these events throughout the distributed system. The monitoring infrastructure system components are distributed among several network nodes in order to minimize the additional overhead to mobile agent applications which might be imposed by deploying the monitoring facilities of *Specto*.

Monitoring Model for Mobile Agents The monitoring model is based on the model introduced in [8] with some modifications which take the monitoring activities of mobile agents into account more accurately. In general, the monitoring model consists of two parts: The generation, and the processing and dissemination of agent state reports within a distributed agent environment.

```
<!-- DTD fo events -->
<!ELEMENT events (preamble, (log | warn | error)*)>
<!ELEMENT preamble (version, timeZone?, datePattern?,
             messageType?, source?, startDate?, info?)>

<!ELEMENT version (#PCDATA)>
<!ELEMENT timeZone (#PCDATA)>
<!ELEMENT datePattern (#PCDATA)>
<!ELEMENT messageType (#PCDATA)>
<!ELEMENT startDate (#PCDATA)>
<!ELEMENT info (#PCDATA)>

<!ELEMENT log (source?, date, msg, param*)>
<!ELEMENT warn (source?, date, msg, param*)>
<!ELEMENT error (source?, date, msg, param*)>

<!ELEMENT source EMPTY>
<!ELEMENT date (#PCDATA)>
```

```
<!ELEMENT msg (#PCDATA)>
<!ELEMENT param EMPTY>


<!ATTLIST log sev      CDATA #IMPLIED
              id          ID #IMPLIED
              fatherID IDREF #IMPLIED>
...

<!ATTLIST error sev      CDATA #IMPLIED
                id          ID #IMPLIED
                fatherID IDREF #IMPLIED>

<!ATTLIST source app   CDATA #REQUIRED
                 class CDATA #IMPLIED
                 IP    CDATA #REQUIRED>

<!ATTLIST param type  CDATA #IMPLIED
                name  CDATA #IMPLIED
                value CDATA #REQUIRED>
```

Fig. 2. Document Type Definition of the *Specto* Markup Language for Monitoring of Distributed Systems

Generation The process of generation of monitoring information of agents addresses the issues of detecting important states of an agent and creating a respective agent state report.

Each lounge and its agents provide information about their current state. This is referred to as agent state information (ASI). A set of ASI forms an agent state report which contains information about an agents' behavior and its influences on its execution environment over an arbitrary period of time. Each state report is encoded as a well-structured XML document [15]. Thus, the generation of ASI is accomplished in an uniform way which does neither favor a particular data format nor the use of special programming or scripting languages. The use of XML as the primary data format for state information of mobile agents enables an agent state report with query capabilities, such as the execution of structured queries to each ASI. Therefore, an agent state report builds an XML-based knowledge repository which holds all relevant information about an agents life cycle. This is referred to as a long-term knowledge repository of agents.

Each agent state report conforms to the *Specto* Markup Language (*SpectoML*). The Document Type Definition of *SpectoML* is shown in Figure 2. It provides three types of monitoring information: *log*, *warn*, and *error*. Each type has a set of attributes such as severity (*sev*) or message identifier (*msg id*). The message text of each monitoring information is uniquely identified by its message identifier (*msg id*) which serves as primary key for retrieving monitoring information within repositories of XML instances. *SpectoML* provides a way to build hierarchical groups of XML instances by using the *fatherID* attribute which is a reference to another monitor information (*id*). As depicted in Figure 2, there are optional parts (attribute list *param* with a triple *name*, *value*, *type*) in the *SpectoML* such as lists of names, values and types, which offers an extensible mechanism to integrate arbitrary kinds of agent-related information to the agent state report.

A *logger* is responsible for the generation of agent state reports. As illustrated in Figure 1, a logger is plugged into the lounge and agent, respectively. From an implementation point of view, a logger is implemented in the programming language the agent environment is using and it is completely decoupled from the distribution infrastructure which is discussed below. The separation of the generation and the dissemination of agent state reports allows a monitoring infrastructure which can be dynamically applied to the core agent environment of *Okeanos* at run-time.

Processing and Dissemination After the process of generation, each agent state report is processed by validating the generated monitoring information and ASI. Then, it is distributed to any interested party. A party in the agent-enabled *Specto* infrastructure can be any agent or agent service which is located at a lounge. In order to enable agents of any kind to retrieve monitoring information of a particular type, the generated monitoring information must be well-structured and a valid XML-document regarding the *SpectoML*.

As shown in Figure 1, a *Cerberus* instance is responsible for contacting and querying a knowledge repository at a lounge. It is designed as a separate software process in order to minimize any dependencies between the *Okeanos* and *Specto* infrastructure and to maximize the reliability of each system environment.

The primary distribution of an agent state report is performed by a light-weight, message-oriented middleware component, called *Mitto*. *Mitto* establishes a global event notification system which consists of an arbitrary number of so-called event channels. It is characterized by its publish-subscribe communication paradigm as opposed to the traditional request-response model. As depicted in Figure 1 the publisher (*Cerberus*) and subscriber (*Event Promoter Agent*) of agent state reports are completely decoupled from each other. In order to participate in such a multicast communication scenario, a subscriber has to register for a particular event channel by providing to *Mitto* a callback handle and a subscription filter.

Each agent subscriber describes the interest in receiving particular agent state reports by using the agent subscription language (ASL). ASL allows an agent to define a set of ASIs which the agent is interested in receiving. For instance, an Event Promoter Agent wants to be informed about all ASI at a particular lounge with regard to the creation of a mobile agent at this lounge. Therefore all elements and attributes of *SpectoML* (Figure 2) can be used to classify the Event Promoter Agent's interest. From a conceptual point of view, the set of ASIs, which are delivered to the Event Promoter Agent, forms just another XML-based knowledge repository which is now related to the subscribing agent.

2.3 Exemplary Application: A Distributed Satisfiability Prover

We use the parallel satisfiability (SAT) prover for boolean logic described in [9,10] where mobile agents organize the distribution of a given task. We contrast this system to our approach which incorporates knowledge of the global system state to gain surplus value, that is in our case, improved workload, balance, and throughput.

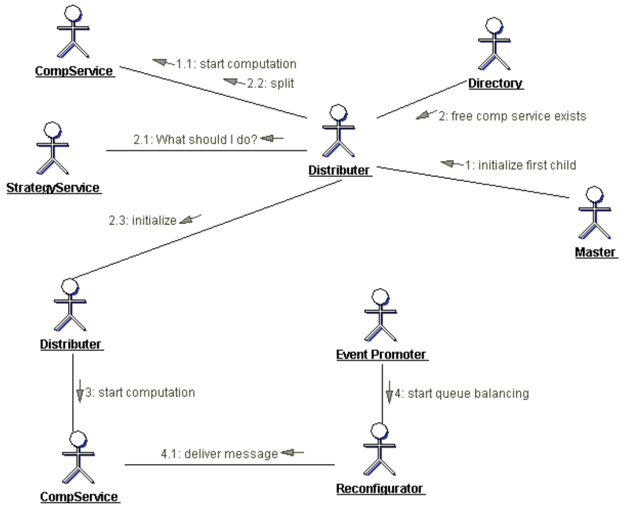


Fig. 3. Collaboration Diagram of the SAT Prover

The SAT problem asks whether or not a Boolean formula has a model, or, alternatively, whether or not a set of Boolean constrains has a solution. The *Davis Putnam* algorithm [2] is used to perform an optimized search for satisfying assignments which leads to an unbalanced search tree. Therefore, dividing the tree at its root produces two subtrees which are different in size. Furthermore, their size cannot be predicted in advance. Consequential, the problem can be divided but it is not predictable how long it will take to search each subtree. This parallelizable but *non-deterministic* problem exemplifies many irregular problems where workload has to be distributed without being able to predict the decomposition.

Agent Components of the SAT Prover The *Master Agent* initiates the first *Distributer Agent* with its SAT problem (Figure 3(1)). That agent looks for a *Computation Service* on its own and starts the computation by transferring the computation parameters (Figure 3(1.1)).

The parallel SAT prover consists of four classes of agents which are described in greater detail in the following paragraphs.

Computation Service Agent. These are stationary service agents one of which is located at every lounge that joins the computation cluster. The Computation Service wraps the underlying algorithm which is implemented in C++ [9] and interfaced by using the Java Native Interface (JNI).

Master Agent. The *Master Agent* launches the task by creating the *Distributer Agent* (see below) and initializing it with the given problem(Figure 3(1)). During its lifetime, the *Master* recollects all subtasks and checks on their results in order to approve consistency. Finally, it displays the result.

Distributer Agent. Agents of this class divide the given task in sub problems if necessary and represent their transporters. If a *Distributer* detects unused nodes, it asks the *Strategy Service* (see below) whether to divide its job or not(Figure 3(2.1)). Receiving the advice to split up, the computation is stopped and the part which has not been reached yet is divided(Figure 3(2.2)). One portion is transferred to the *Computation Service*(Figure 3(1.1)), the other is handed to a newly created *Distributer* (Figure 3(2.3)) which has to look for an available computation node by itself. Of course, multiple *Distributers* can arrive at a *Computation Service*, previously declared available. If so, the queue is managed by the *Computation Service* and the tasks are handled one by one. Finally, after completing their work, the *Distributers* return to the *Master* and deliver the result of their respective task. Due to the variety and complexity of duties, the previously mentioned integrated expert system shell JESS is used. Therefore, different sets of rules apply to respective domains such as the reaction on context changes, splitting up, finding available computation nodes, etc.

Strategy Service. This stationary service agent gives *Distributer Agents* hints and suggestions whether to split up their computational task or not. This decision is based on the local service directory which is the only view towards the global system state. The actual implementation advises to divide if any available computation node exists. Due to the limited amount of information about agent configurations at other remote lounges, the capability of the *Strategy Service* to suggest a global optimal splitting strategy is rather restricted. In the following section, these shortcomings are addressed by deploying distributed state information based on long-term knowledge repositories.

2.4 Deploying Long-Term XML-Based Knowledge Repositories

For the deployment of knowledge repositories two additional kinds of agents are incorporated into *Okeanos*:

Event Promoter Agent. This kind of agents subscribes to the notification middleware *Mitto* by defining a set of filters. Each filter determines the type and attributes of agent states the *Event Promoter* is interested in receiving from remote lounges. For instance, the *Event Promoter Agent* subscribes for the number of currently waiting *Distributer Agents* in the waiting queues at the remote lounges. Given the information about all queues, this agent tries to relocate all waiting *Distributers* in order to balance the lengths of all queues. The quantum of agents transferred from the longest to the shortest queue with length greater than 0 is evaluated according to the formula shown in Figure 4.

Once this amount is determined and greater than zero, a **Reconfigurator Agent** gets created and initialized with the determined parameters (Figure 3(4)). It is then disseminated to inform the computation service with the longest queue about this decision (Figure 3(4.1)).

The computation service itself dequeues the requested number of *Distributers* and advices them to migrate to the transmitted service with the shortest list of waiting agents. This way, the amount of waiting *Distributers* gets balanced globally.

$$med = \frac{\sum_{i=1}^{n} queue_i}{n} \tag{1}$$

$$\#agents = \lfloor \min(med - queue_{min}, queue_{max} - med) \rfloor \tag{2}$$

$$total\ distance = \sum_{i=1}^{n} \lfloor |queue_i - med| \rfloor \tag{3}$$

Fig. 4. The *Event Promoter Agent* moves the evaluated amount of Distributers from the longest to the shortest queue (2). The total agent distance where n is the number of lounges and $queue_i$ is the number of waiting agents at lounge i (3)

The robustness of the application is in no way affected by the reorganization of *Distributer Agents*. The application will terminate correctly even if the *Event Promoter Agent* might be destroyed eventually.

3 Empirical Results

We have used our departmental infrastructure for several distributed computations of the SAT prover. For the realization of performance measurements we have selected one benchmark (*dubois23*) from the publicly available[1] DIMACS benchmark suite for SAT provers. The chosen benchmark exhibits a highly irregular search splitting behavior. The processing time for a given search sub tree is not predictable and therefore an optimal problem solving plan cannot be determined in advance. Due to this fact each run of this benchmark within the same distributed environment produces slightly different run-time results. In our previous work [10], we discussed this issue of irregularity on a broader base of benchmarks and configurations. However, in this paper we focus on the analysis of the benefits emerging from system-wide long-term state information.

The measurements were carried out on a heterogeneous pool of machines containing 2 Pentium II (300MHz, 450MHz), 10 SUN Ultra workstations and 2 SUN enterprise servers. Each benchmark is performed using the Java Virtual Machine which comes with the JDK 1.3.

The SAT prover is integrated into *Okeanos* as a computation service by interfacing to its C++ implementation using JNI. The libraries were compiled for each hardware platform in advance.

The figures presented in this section are generated by deploying the knowledge repositories residing at each lounge which are generated during application run-time. The size of these XML-based repositories are in a range of 3 to 15 MByte for an execution time from about 15 up to 90 minutes depending on the distributed computation node configuration. For statistical purposes, a set of queries is sent to each repository to retrieve the global system state during the distributed computation, right after the application has finished.

[1] `ftp://dimacs.rutgers.edu/pub/challenge/sat/benchmarks/cnf/`

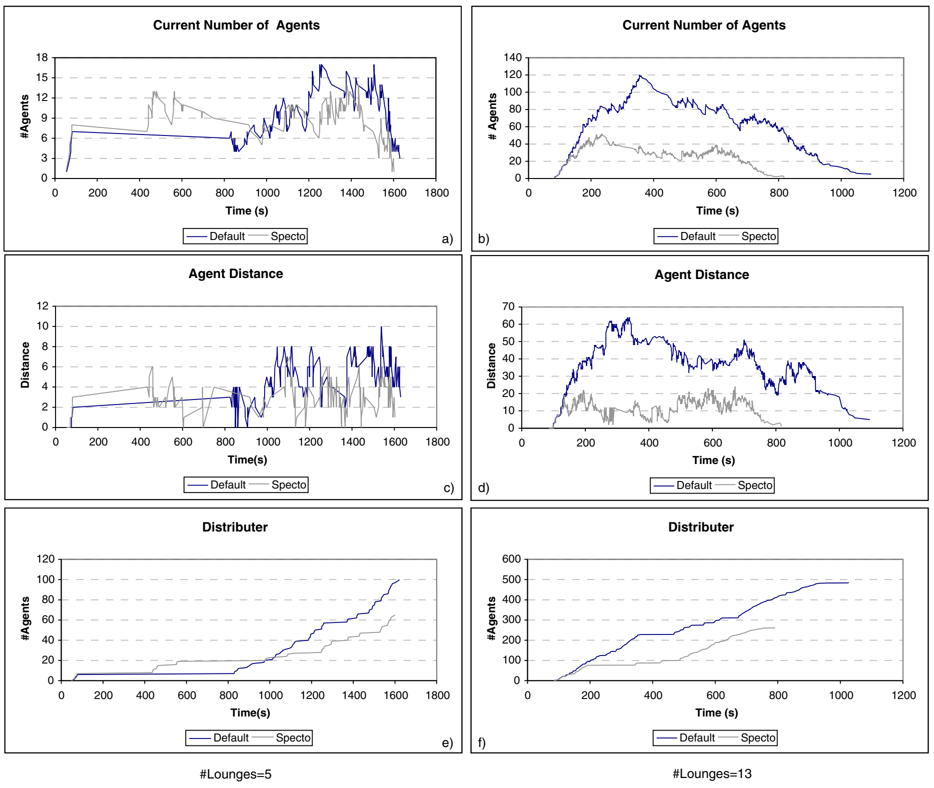


Fig. 5. SAT prover runs of *dubois23* by using the autonomous distribution and processing of mobile *Okeanos* agents (*Default*) and by additionally deploying *Specto*'s monitoring infrastructure (*Specto*). The total participating lounges vary from 5 (left column of figure) to 13 (right column of figure)

Figure 5 shows the results for the benchmark for 5 and 13 computation nodes (lounges). Each benchmark runs using a processing strategy which takes only local information into account and is marked as *Default*. A more sophisticated processing strategy which makes use of the long-term knowledge repositories is marked as *Specto*.

Current Number of Agents In general by using *Specto* the current number of processing agents waiting in computation service queues is reduced (Figure 5b). However, by only deploying small numbers of computation nodes the difference of the number of waiting agents between both strategies is rather close.

As depicted in Figure 5a, *Specto*'s graph can even be worse due to an adverse distribution of sub tasks which causes an early split of tasks. Subsequently, the graphs cross and the default implementation leads to the use of more agents again.

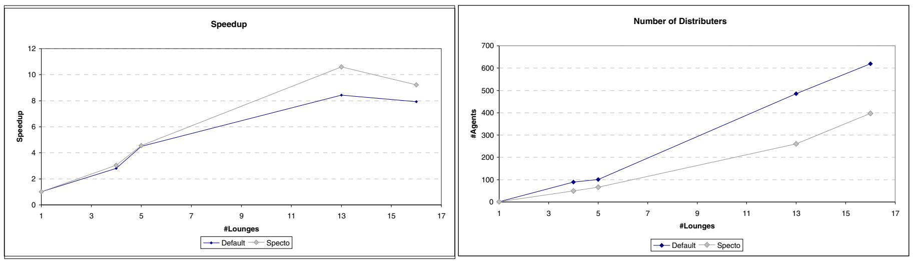


Fig. 6. Distributed search of *dubois23.cnf* without (*Default*) and with additionally deploying *Specto*'s monitoring infrastructure (*Specto*). All results regarding the varying number of lounges

Agent Distance The agent distance (AD) is an indicator for the overall balance of the workload at all lounges (Figure 4). The optimal AD is 0 indicating that each lounge hosts the same number of waiting agents.

If one computation unit runs out of tasks, all other *Distributer Agents* are notified about that fact (see *directory service* in 2.1). The first *Distributers* in the queue whose tasks are currently worked on try to split them up and send the newly created sub problems to this available lounge. This splitting process is expensive because all computations have to be stopped in order to be split and the spawned subtasks must be transferred. To avoid this situation as long as possible, the waiting queues should be balanced, and therefore AD should approach 0. As shown in Figure 5d the *Specto* strategy improves the equipartition of waiting agents significantly. In contrast, the distribution introduced by the *Default* strategy is more than twice as high. However, the positive effect of distributing the processing agents uniformly among the lounges is less efficient if the number of participating lounges is small (Figure 5c). This stems from the fact that with a smaller number of lounges the split-overhead is much smaller, because much fewer tasks are divided and have to be transferred.

The Overall Number of Processing Agents Finally, the comparison of the two strategies shows that the *Specto* strategy using long-term knowledge of the overall system state contributes to the performance of an irregular, distributed computation better if there are more participating computation nodes. In both Figures 5e and 5f, the total number of agents using the advanced approach is less than 65% of the default system at the end of execution time.

The bigger the number of lounges in Figure 5f becomes, a shorter total execution time is apparent. The only slight gain in time in Figure 5e is founded in the awkward distribution in the first half of the execution and in the smaller split overhead.

Figure 6a depicts the speedup approaching nearly the linear maximum speedup when 5 lounges are used (the absolute time values have to be considered with respect to the heterogeneous hardware and software environment used for these measurements). Incorporating 13 computation units, the slope gets smaller with the *Specto*-approach being slightly better, but is still increasing.

The system applying 16 nodes shows less performance than with 13 nodes for both approaches which is due to the now top communication overhead. Nevertheless, the load balancing system still works better than the default system.

The number of *Distributer Agents* increases roughly linearly with a growing number of lounges (Figure 6b) where the slope of the *Specto*-approach is significantly lower. The difference of about 200 *Distributer Agents* is filled by about the same amount of *Reconfigurator Agents* used to balance the waiting queues but these agents, being very simple have much less creation and migration overhead.

These results indicate, that this benchmark (*dubois23*) cannot be computed faster with even more lounges. However, a longer running benchmark could perhaps achieve its maximum performance with more nodes but it is not the intention of this work to reason about absolute performance gain rather than on pointing out the surplus value of long-term global system state knowledge. With these results, it can be a valuable approach for long-running computational tasks within a large-scale distributed system, to take long-term global information into account. However, it is clear that the simple load-balancing strategy can be further improved by taking a set of different balancing strategies at the same time into account, for instance.

4 Discussion

In this section, issues in designing and using long-term knowledge repositories for mobile agent applications are discussed. An XML-based knowledge repository as described in Section 2.2 offers agents in general and mobile agents in particular several benefits:

(1) State information of each agent and its respective agent execution environment can be made persistent in an uniform way. All kinds of system components are enabled to search for particular state information in a well-defined, structural manner which is similar to common database management facilities. The subscription language allows agents to set up structured queries easily. The *Event Promoter Agent* described in Section 2.4 subscribes for a particular set of state information which forms the basis for its own knowledge repository during the distributed execution of the SAT prover application. Its knowledge repository reflects the overall distributed system state which all of its load-balancing strategies are based on.

(2) The use of XML as a general data description format does neither favor any dedicated data format nor any particular agent programming language, as long as the programming language in mind offers a means to generate XML documents. Therefore, knowledge repositories based on XML documents provides an interoperable platform for different agent architectures to cooperate with each other. The *Event Promoter Agent* is implemented in Java. However, several other dedicated load-balancing agents might be transparently integrated into *Okeanos* and *Specto* respectively. These agents or services could be imple-

mented in any programming language which offer an application programming interface to generate XML documents and provide a communication facility for interacting with Java-based agents (e.g. by using JNI or plain sockets).

(3) Cooperation among agents can be accomplished by using XML-based knowledge repositories. The subscription and disseminating mechanism of *Specto* and *Mitto* allow agents to retrieve state information of other agents residing on remote lounges rather easily. However, since the monitoring infrastructure is completely separated from the agent environment, an agent does not rely on the monitoring facilities at all. Therefore, *Specto* can be dynamically applied to *Okeanos* during run-time improving the flexibility and extensibility of the agent system. In our measurements the monitoring infrastructure and the *Event Promoter Agents* are started together with all other computational lounges in order to get an accurate view of the run-time behavior of the agent environment. This approach is only due to practical considerations for the study of empirical results. Conceptually, there is no restriction about plugging the monitoring infrastructure into the agent execution environment during run-time.

(4) Despite the cooperation capabilities, XML-based knowledge repositories can provide the necessary information for all kinds of agent-related system components to adapt to current environmental conditions. Since mobile and autonomous agent infrastructures are characterized by their high degree of reactivity, the overall system has to be able to grow and adapt to new environmental conditions without direct reprogramming of agents and their execution environments. For this reason, agents and lounges should be able to sense the current environment by deploying long-term knowledge repositories without trading off the autonomy of a mobile agent (see (3)).

(5) A XML-based knowledge repository is designed as a lightweight system component which can easily be installed, accessed and even moved around within the distributed agent environment. Especially the capability to make an agent's knowledge repository mobile, makes it an perfect candidate to store all state information regarding a mobile agent's execution over its entire life cycle. In contrast to a centralized approach to the design of an agent knowledge repository, the mobile and lightweight approach benefits from the advantages of using mobile agents in general, for instance such as reduced communication overhead between remote and local agents.

However, there are some issues which have to be carefully taken into consideration by establishing and deploying long-term XML-based knowledge repositories:

(1) The volume of the data within a knowledge repository can get rather large and become a limiting factor in the overall agent environment. Due to the human-readable data format of XML documents, the size of the knowledge repository can be decreased by an order of magnitude by using several compression techniques. In our empirical studies (see Section 3) the overall number of an agent's state information ranges from 10,000 to over 157,000 for a distributed scientific application depending on the number of participating computation engines (lounges). For long-running agent-enabled scientific applications the type

and the number of an agent state information has to be carefully taken into consideration.

(2) During long-running and highly distributed agent applications the communication overhead between each lounge (e.g. *Cerberus*) and *Mitto* might be critical. Therefore, *Mitto* provides a so-called transparent stream multiplexing mechanism which offers several kinds of raw communication streams above Remote Method Invocation. Thus, a *Cerberus* instance can use the stream which compresses all data before it is transmitted to *Mitto*. In addition, several buffering techniques can be used to reduce the number of remote calls between a *Cerberus* and *Mitto* (see also (1)).

5 Related Work

MATS [6] is a mobile agent system for distributed processing. It is based on collections of agents which form teams to solve distributed tasks. The user may be forced to structure the problem in such a way that it can be easily broken into a set of co-operating tasks, whereas in *Okeanos* it is the responsibility of the autonomous agent to figure out an appropriate plan to distribute the potentially irregular tasks. The granularity of distribution in *Okeanos* is directly related to each agent's strategy which is determined by applying techniques from the area of artificial intelligence.

The database-based approach to monitoring complex and distributed systems is introduced by Snodgrass in [13]. It uses a relational data format for capturing relevant monitoring information. In contrast, *Specto* take use of lightweight and mobile XML-based knowledge repositories which provide more accurate monitoring information within an mobile agent environment.

Agent coordination technologies like MARS [1] emerge using Linda-like tuple spaces. However, these systems pursue a different approach by deploying pattern matching techniques contrary to our XML-based database approach. The tuple spaces focus on short term interaction schemes whereas our approach is targeted to persistent distributed overall system state.

6 Conclusion and Future Work

In this paper, we introduced the concept of XML-based long-term knowledge repositories which enable mobile and autonomous agents to cooperate effectively by deploying the overall distributed system state. We evaluated our approach by using an agent-based distributed problem solving environment which decomposes a given irregular problem into a set of parallizable but unpredictable sub problems. Each of these sub problems is processed in parallel and independently by autonomous and mobile agents within a distributed agent infrastructure. The empirical study, based on several distributed environmental setups and a heterogeneous pool of hardware and software components, shows that the benefits of XML-based knowledge repositories for the deployment of distributed system states for mobile agents are evident. However, future research is needed and will focus onto the design of more sophisticated cooperation strategies based on knowledge repositories.

References

1. G. Cabri, L. Leonardi, and F. Zambonelli. XML dataspaces for mobile agent coordination. In *Proceedings of the 2000 ACM Symposium on Applied Computing*, Mar. 2000. 93
2. M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. In *Journal of the ACM*, volume 7, pages 201–215, 1960. 86
3. T. Finn, Y. Labrou, and J. Mayfield. KQML as an Agent Communication Language. In J. Bradshaw, editor, *Software Agents*, pages 291–316. MIT Press, 1997. 82
4. E. Friedman-Hill. Jess, The Java Expert System Shell. Available at the URL: http://herzberg.ca.sandia.gov/jess/, 1999. 82
5. A. Fugetta, G. Picco, and G. Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998. 81
6. M. Ghanea-Hercock, J. Collis, and D. Ndumu. Co-operating mobile agents for distributed parallel processing. In O. Etzioni, J. Müller, and J. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*. ACM Press, 1999. 93
7. R. Johnson and B. Foote. Designing Reusable Classes. *Object-Oriented Programming*, 1(2):22–35, 1988. 82
8. J. Joyce, G. Lomow, K. Slind, and B. Unger. Monitoring distributed systems. *ACM Transactions on Computer Systems*, 5(2):121–150, Feb. 1987. 83
9. W. Küchlin and C. Sinz. Proving Consistency Assertions for Automotive Product Data Management. In I. P. Gent and T. Walsh, editors, *Journal of Automated Reasoning*. Kluwer Academic Publishers, 2000. 85, 86
10. R.-D. Schimkat, W. Blochinger, C. Sinz, M. Friedrich, and W. Küchlin. A service-based agent framework for distributed symbolic computation. In M. Bubak, R. Williams, H. Afsarmanesh, and B. Hertzberger, editors, *Proceedings of the 8th International Conference on High Performance Computing and Networking Europe (HPCN'00)*, volume 1823, pages 644–656, Amsterdam, Netherlands, May 2000. Springer-Verlag, Berlin. 82, 85, 88
11. R.-D. Schimkat, M. Häusser, W. Küchlin, and R. Krautter. Web application middleware to support XML-based monitoring in distributed systems. In N. Debnath, editor, *Proceedings of 13th International Conference on Computer and Applications in Industry and Engineering (CAINE 2000)*, pages 203–207, Hawaii, USA, Nov. 2000. International Society for Computers and Their Applications. 82, 83
12. R.-D. Schimkat, S. Müller, W. Küchlin, and R. Krautter. A lightweight, message-oriented application server for the WWW. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, editors, *Proceedings of the 15th ACM Symposium on Applied Computing (SAC 2000)*, pages 934–941, Como, Italy, Mar. 2000. 82
13. R. T. Snodgrass. Relational approach to monitoring complex systems. *ACM Transactions on Computer Systems*, 6(2):157–196, 1988. 93
14. J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A note on distributed computing. Technical Report TR-94-29, SUN Microsystems Laboratories, Nov 1994. 81
15. World Wide Web Consortium (W3C), http://www.w3.org/TR/REC-xml. *Extensible Markup Language (XML) 1.0*, Oct. 2000. 84

Cooperative Meeting Scheduling among Agents Based on Multiple Negotiations

Toramatsu Shintani[1] and Takayuki Ito[2]

[1] Intelligence and Computer Science, Nagoya Institute of Technology,
Gokiso, Showa-ku, Nagoya, Aichi 466-8555, Japan.
tora@ics.nitech.ac.jp

[2] Center for Knowledge Science, Japan Advanced Institute of Science and Technology,
1-1 Asahidai, Tatsunokuchi-machi, Nomi-gun, Ishikawa 923-1292, Japan.
itota@jaist.ac.jp

Abstract. We present a method for multi-agent negotiation for implementing a distributed meeting scheduler. In this system, an agent is assigned to an user who plans to schedule private events. Each agent negotiates with other agents in order to make a public schedule by referring user's private schedules and preferences. We propose a new persuasion method for multi-agent negotiation for reflecting private preferences. We call this method the **Multiple Negotiations**. The multiple negotiations can effectively facilitate reaching an agreement among agents. In the multiple negotiations, agents conduct all combinations of negotiation in which each agent has an opportunity for persuading the others. In addition, we propose an effective preference revision mechanism based on the multi attribute utility theory. The mechanism facilitates negotiation among agents and improves quality of the multiple negotiations. We have implemented a distributed meeting scheduler to show how effectively the multiple negotiations can be used. The result shows that the multiple negotiations are effective in supporting group decision-making for scheduling a meeting.

1 Introduction

In social decision making, we need to clarify a trade-off between "reaching a consensus" and "maximizing own expected payoffs". During the past decades, several solutions to the coalition formation problem have been presented by researchers in DAI(Distributed Artificial Intelligence) [10]. Cooperation among autonomous agents may be mutually beneficial even if membership in the coalition may maximize a personal outcome. In Japanese social decision making, some of solutions for the coalition can be based on persuasion [8]. In agent negotiations, the persuasion mechanism can be defined as follows [5]. In a persuasion mechanism between agents, an agent who persuades another agent is called a **persuader**, and an agent who is persuaded by a persuader is called a **compromiser**. The persuasion mechanism can be outlined as follows: (**1. Proposal**)

© Springer-Verlag Berlin Heidelberg 2001

A persuader sends a proposal to a compromiser in order to reach an agreement. (**2. Preference revision**) The compromiser receives the proposal. If the compromiser is able to accept the proposal, he needs not revise his preferences. If he is unable to accept the proposal, the compromiser tries to revise his preference in order to accept the proposal. (**3. Reply**) If the compromiser is able to accept the proposal as a result of the utility revision, he replies with an agreement message. If not, the compromiser replies with a reject message.

The negotiation problem in a multi-agent system has been investigated very actively [15]. A distributed scheduler for a meeting [13] is an application of multi-agent systems. In this paper, we present an architecture for multi-agent negotiation for implementing a distributed meeting scheduler. The meeting scheduler plans schedules (such as a meeting) based on a multi-agent system. We propose an effective preference revision mechanism based on MAUT(Multi Attribute Utility Theory) [7]. In general, MAUT handles problems in which outcomes are characterized by two or more attributes. For example, purchasing a new car requires consideration of a price, a color, a type (sporty, luxury, etc), and so on. In MAUT, for an alternative C_i, there exist the attributes $X_1, X_2, \ldots, X_n$ and their values $x_1(C_i), x_2(C_i), \ldots, x_n(C_i)$. We can represent the utility $u(C_i)$ for the alternative C_i as $u(C_i) = f(f_1(x_1(C_i)), \ldots, f_n(x_n(C_i)))$, where f is a certain function. We can select several options with respect to f according to the application area. In our system, we select the AHP [11] method for calculating user's utility. Based on the above utility, each agents has a preference. According to von Neumann-Morgenstern, we define agent's preference as follows: $C_i \succ C_j \Leftrightarrow u(C_i) > u(C_j)$ and $C_i \sim C_j \Leftrightarrow u(C_i) = u(C_j)$. $C_i \succ C_j$ means that an agent prefers C_i to C_j. $C_i \sim C_j$ means that the agent is indifferent between C_i and C_j.

Multi-agent meeting scheduling has been a current topic of research and has been studied widely. Sen and Durfee [12] has been focused on solving meeting scheduling problem using a central host agent. However, user preferences are not taken into account. Haynes, et al. has developed a system [4] in which agents can schedule meetings within users' preferences. In this system, user's preference is represented by values with a threshold. Agents try to adjust values under a threshold which means a degree to which a value can or cannot be compromised. As a novel approach, Garrido and Sycara [3] has been focused on decentralized meeting scheduling with user preferences taking into account. Here the agents can negotiate and relax their constraints to find and reach agreements on schedules with high joint utility. In our approach, agents negotiate and reach agreements based on users' subjective private preferences without a concept of joint utility. In terms of persuasion among agents, the argumentation-based persuasion [14] has been focused in past few years. Parsons et. al [9] proposed the logic-based argumentation framework for negotiation among agents. In this framework, persuasion among agents is described logically. The difference between their approach and our approach is that our persuasion is based on user's utility and preference.

The aim of this paper is to present a new architecture for multi-agent negotiation for realizing a negotiation algorithm in a distributed meeting scheduler based on the multiple negotiations mechanism. The paper consists of five sections. In Section 2, we show the architecture of our system. In Section 3, we present the negotiation process based on private preferences. In Section 4, we show an example and discuss the results of the multiple negotiations. Some concluding remarks are presented in Section 5.

2 The Distributed Meeting Scheduler

2.1 The Agent-Based Architecture

In the meeting scheduler, an agent is assigned to an user who plans private schedules and events in a calendar. A supervisor manages to plan schedules and events. We call the supervisor the host user. An agent assigned to a host user is called a host agent. Candidates invited to an event are called attendee users. An agent assigned to an attendee user is called an attendee agent. In order to schedule a public event, a host agent negotiates with other attendee agents by using users' private schedules and preferences. Agents can reach an agreement about a public event. The point is that agents need to clarify a trade-off between an agreement about scheduling a public event and users' private schedules. We improve the trade-off by using a **MAUT-based persuasion process** and by employing the multiple negotiations. The MAUT-based persuasion process and the multiple negotiations we proposed here can facilitate reaching an agreement among agents effectively and reflect the users' preferences in a negotiation process among agents.

2.2 Cooperative Distributed Scheduling Process

Fig.1 shows a scheduling process we used in a distributed meeting scheduler. Before scheduling meetings, each user has inputted their private schedules into their own calendar. The meeting scheduler has four phases. In the **first** phase, an user requests for scheduling a meeting. The user becomes a host user. In the **second** phase, a host user decides the size and length of a meeting . The length of a meeting is the number of hours. This length means how long a meeting is held. The size of a meeting is the number of attendee. A host user choices time intervals that satisfy the length of the meeting. We call these time intervals the candidate time intervals. A host agent proposes candidate time intervals at the beginning of the third phase. In the **third phase**, each user subjectively judges the candidate intervals. User's judgements are quantified into the MAUT-based preference described in Section 1. The details of this phase are described in the next section. In the **forth phase**, by using a persuasion protocol, agents negotiate with other agents using users' private preferences (i.e., schedules and attributes) of the meeting that are decided by a host user. In this phase, the multiple negotiations are conducted by agents. The details of this phase are

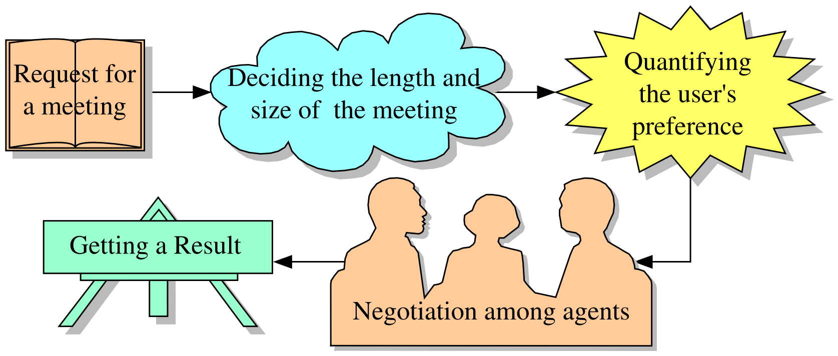

Request for a meeting
Deciding the length and size of the meeting
Quantifying the user's preference
Negotiation among agents
Getting a Result

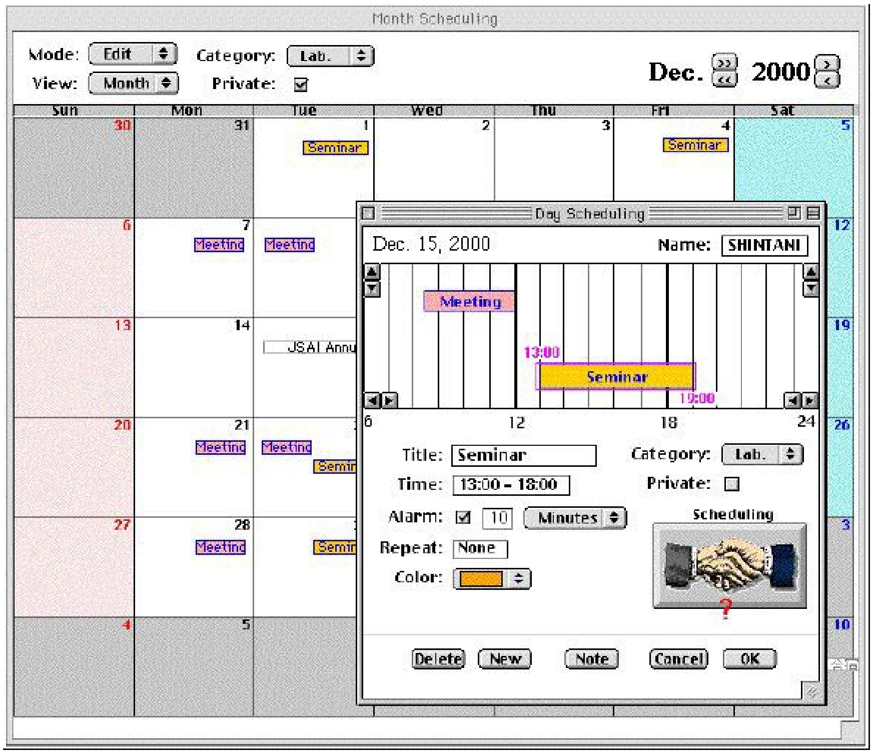
Fig. 1. The Distributed Meeting Scheduling Process

described in Section 4. In the **fifth phase**, a host agent broadcasts the result of the agent negotiation.

If agents can not reach a consensus, the result of a negotiation is that a schedule for a meeting proposed by a host user is rejected. If agents reach a consensus, the result is that a schedule for the meeting will be held as a public schedule.

Fig.2 shows the user interface of the distributed meeting scheduler. The system has been implemented by using MiLog (**M**obile **i**ntelligent agents using **Log**ic programming) [2]. MiLog is implemented by using Pure Java. To realize an efficient mobile agent development environment, Milog provides a hybrid programming environment in which a mobile agent can be designed by logic programming and Java programming. In MiLog, we can create mobile agents that have an anytime migration mechanism, web-service/access functions. Anytime migration enables agents to infer during migration from one computer to another. A MiLog agent can behave as a CGI program and can access to other web servers via HTTP protocol. Further, MiLog provides user interface development tools, iML and MiPage. Using iML, users can create MiLog agents that has Java-based user interfaces. MiPage enables users to create web-based user interface for MiLog agents.

The background window in Fig.2 shows a calendar. In the calendar, an user can get information that several private and public events are scheduled. The bottom right window in Fig. 2 shows a window with the button named "Scheduling" for requesting a new meeting in which an user can define attributes of an event that are a time interval, an alarm checking, a category of the event, a private-public, a note for event details, and so on. If an user schedule a public event (e.g., a meeting) and push the "Scheduling" button, the distributed meeting scheduling process will be started. This calendar has been developed by using iML.


Mode: Edit
Category: Lab.
View: Month
Private: ☑
Dec. 2000

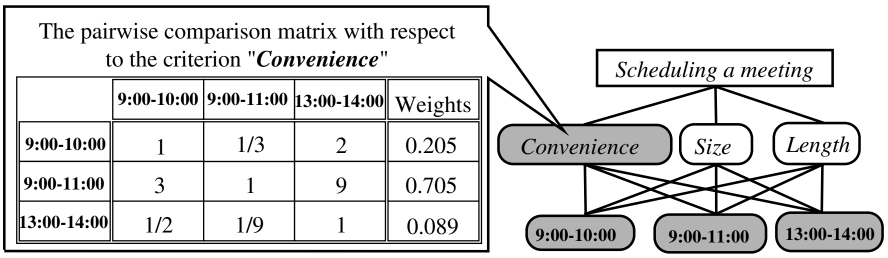
Dec. 15, 2000
Name: SHINTANI
Title: Seminar
Category: Lab.
Time: 13:00 - 18:00
Private: ☐
Alarm: ☑ 10 Minutes
Repeat: None

Fig. 2. User Interface of the Distributed Meeting Scheduler

2.3 Quantifying User's Preference

In order to measure a subjective multiple attribute preference of an user, we employ the AHP-based quantifying method[6]. The AHP enables users to input their preference intuitutively and systematically.

Figure 3 shows a typical hierarchy and a pairwise comparison matrix. In the AHP, users divide a problem into a hierarchy that consists of a goal, criteria (and possibly sub-criteria), and alternatives. For example, in Figure 3 the goal (*Scheduling a meeting*), the overall objective, is decomposed into criteria *Convenience*, *Size*, and *Length*. The alternatives are the candidate time intervals $9:00-10:00$, $9:00-11:00$, and $13:00-14:00$. The features of the decision hierarchy are as follows: (I)Each criterion exists at the same level is independent. (II)Each level of the hierarchy is independent. The judgement of the pairwise comparison between factors (in Figure 3, alternatives $9:00-10:00$, $9:00-11:00$, and $13:00-14:00$) on a certain level is made with respect to the criterion that is a factor (in Figure 3, the criterion *Convenience*) on the upper level. By interpreting a set of judgement values as a matrix (top left of Figure 3), the weights (i.e., measurement of criteria) of factors are derived from the matrix by using an eigenvector approach. To put it more concretely, the weights of each


The pairwise comparison matrix with respect to the criterion "***Convenience***"

	9:00-10:00	9:00-11:00	13:00-14:00	Weights
9:00-10:00	1	1/3	2	0.205
9:00-11:00	3	1	9	0.705
13:00-14:00	1/2	1/9	1	0.089

Fig. 3. Constructing a Decision Hierarchy

factor are derived as the eigen-vector for the max eigen-value of the pairwise comparison matrix. As a whole hierarchy, the weights of the alternatives can be recalculated by composing the weights of the criteria.

Each agent keeps the user's AHP hierarchy as a multi-attribute preference. The term **criterion** means the **attribute** in MAUT. In the above example, the utility for the alternative $9:00-10:00$ can be defined based on the attributes *Convenience*, *Size*, and *Length*. Namely,$u(A_1) = f(f_{C_1}(C_1(A_1)), f_{C_2}(C_2(A_1)), f_{C_3}(C_3(A_1)))$ where f, f_{C_1}, f_{C_2}, and f_{C_3} are provided as the weight calculating functions in the AHP, A_1 represents the alternative $9:00-10:00$, and $C_1, C_2, andC_3$ represent criteria *Convenience*,*Size*, and *Length*, respectively.

3 Reaching a Consensus

3.1 The Multiple Negotiations among Agents

In our previous work [5], agents reach an agreement by using a tournament mechanism. A tournament mechanism consists of several persuasion mechanisms. A persuasion mechanism consists of a persuader and a compromiser. In a persuasion mechanism, a persuader is decided randomly. Therefore, in a tournament mechanism, there are several agents who can not have any opportunity for persuading the other agents. Namely, although a tournament mechanism can facilitate "reaching a consensus", it can not maximize "own expected payoffs". Therefore, in this paper, we propose the multiple negotiations in order to maximize user's own expected payoffs. In the multiple negotiations, agents conduct all combinations of negotiation in which each agent has an opportunity for persuading the others.

In order to get a result which can be agreeable to users, we propose a new persuasion method for multi-agent negotiation for reflecting private preferences. We call the method the **multiple negotiations**. In the multiple negotiations, each agent has an opportunity for persuading the others by conducting all patterns of negotiation. For example, if there exist eight agents, eight patterns of negotiation are conducted. In a pattern of negotiation, a certain agent persuades the other agents. Therefore, each agent can persuade the other agents in the multiple negotiations. Furthermore, since agents exchange information on processes of negotiation in the multiple negotiations, agents can reach a consensus more effectively. In order to reach a consensus effectively in a pattern of negotiation, agents try to persuade the other agents.

The multiple negotiations can be modeled by a 4-tuple $< N, C, P, E >$, where,

- N is a set of agents, $N = \{1, \ldots, i, \ldots, n\}$.
- C is a set of clones of agents, $C = \{C_1, \ldots, C_i, \ldots, C_n\}$, and $C_i = \{c_{i1}, \ldots, c_{in}\}$ (the clone c_{ik} means that c_{ik} is the agent i's k-th clone).
- P is a set of patterns of negotiation, $P = \{p_1, \ldots, p_i \ldots, p_n\}$. The p_i is consisted of a_i and $c_{1i}, \ldots, c_{ji}, \ldots, c_{ni}, j \neq i$ (a pattern p_i is consisted of agent i and the other agent's i-th clones). In one pattern, the agent a_i and the clones negotiate with each other. In our system, the agent a_i persuade the clones.
- E is an evaluation function of the result of negotiation of a pattern, $E(p_i) = \mathcal{R}$. Users can select the best result based on the evaluation function. $\mathcal{R}$ means a certain real number.

In the multiple negotiations, all patterns of negotiation are conducted. In order to facilitate the multiple negotiations, all patterns of negotiation should be conducted concurrently. The cloning technique for mobile agents enables us to realize concurrent negotiation processes for the multiple negotiation. We implement agents as MiLog's mobile agents. MiLog's mobile agents have the anytime migration mechanism that enables them to infer during transportation. Since agent's clones are also MiLog agents, they can infer even during migration. The reason why we implement agents as mobile agents is that in the multiple negotiations, agents need to conduct a lot of patterns of negotiation. Since it is hard to conduct all patterns on one computer, agents try to conduct each pattern by using distributed computers. This can distribute computational load efficiently. Traditional concurrent techniques or parallel techniques also enable us to implement the multiple negotiations mechanism. We however need a lot of communications between agents if we employ these traditional techniques.

Fig.4 shows the steps of the multiple negotiations among agents. Initially, there exists one agent on each computer. We call these agents "host-agents." First, each host-agent dispatches his clones to the other computers. These clones are called "member-agents." Second, agents negotiate with each other. In Fig. 4, since there exists four agents, agents conduct four patterns of negotiation, Pattern 1, Pattern 2, Pattern 3, and Pattern 4. In each pattern of negotiation, host-agents persuade the other member-agents. The details of persuasion are described at Section 4.2. Finally, each clone returns to the original host computer in order to report the results of the negotiation.

In the second step, in order to facilitate negotiations, host-agents exchange information on process of negotiation. The information includes which agent accepts or rejects a proposal in persuasion. By using these information, host-agents can effectively reduce work required to persuade member-agents. For example, if

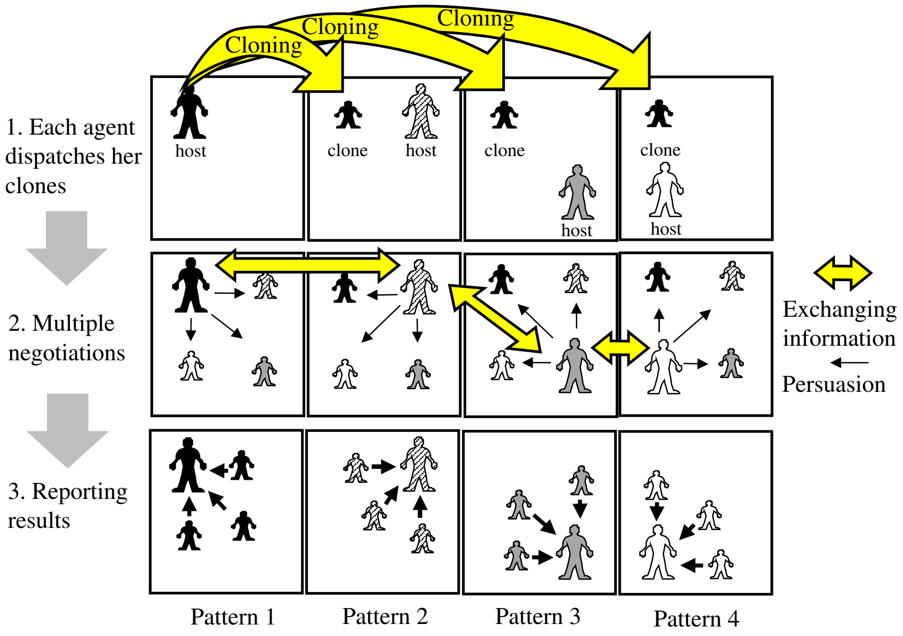


Fig. 4. Multiple Negotiations among Agents

the agent X could not persuade the agent Y's clone with the proposal Z, we can assume that the agent Y's other clones also reject the proposal Z. Therefore, the agent X sends this information (i.e. information on that the agent Y's clone rejected the proposal Z) to the other host-agents. The host-agents, which are received this information, need not to persuade the agent Y's clone. This means that agents can save times to persuade agent Y's clones.

In our system, a host user selects the best result from multiple results of the multiple negotiations. We consider the two points for selecting the best result, which are times required for reaching a consensus and number of agreements. For example, if in a certain pattern, agents can reach a consensus faster than the other patterns, the host user selects this result as the best result.

3.2 The Preference Revision Based on Private Preferences

In the multiple negotiations, agents try to reach a consensus by using the persuasion mechanism. In persuasion, a compromiser tries to revise his preference so that his most preferable alternative is the same as a persuader's most preferable alternative. The most preferable alternative have the highest utility. For example, if a compromiser's preference is $A_1 \succ A_2 \succ A_3$ and a persuader's most preferable alternative is A_2, the compromiser tries to change the preference to $A_2 \succ A_1 \succ A_3$ or $A_2 \succ A_3 \succ A_1$, in which A_i is an alternative, and the alternative A_2 is the most preferable. In order to change the order as above, agents try

```
INPUT : The persuader's most preferable alternative($PA$) and
        the compromiser's original preference($PreF$)
OUTPUT: $Success$ or $Failure$
Function $PrefRevision(PA, Pref)$
  $PATS := apowersetofattributesforthealternativePA.$;
  $SortedPATS := sortBySize(PATS)$; $Candidates := \phi$; $Solutions := \phi$;
  For each $ATS$ in $SortedPATS$
    $ATS' := IncreaseValues(ATS)$; $Pref' := ReCalculate(Pref, ATS')$;
    $Candidates := Candidates \cup Pref'$;
    If $PA == theMostPreferableAlternative(Pref')$ Then
      $Solutions := Solusions \cup Pref'$;
  If $Solutions$ is not empty Then
    $Pref := selectMinimalPref(Solusions)$;
    return $Success$
  $PATS :=$ a power set of attributes for the Most preferable alternative($MA$);
  $SortedPATS := sortBySize(PATS)$;
  For each $CandidatePref$ in $Candidates$
    For each $ATS$ in $SortedPATS$
      $ATS' := decreaseValue(ATS)$; $Pref' := ReCalculate(CandidatePref, ATS')$;
      If $PA == theMostPreferableAlternative(Pref')$ Then
        $Solutions := Solusions \cup Pref'$;
  If $Solusions$ is not empty Then
    $Pref := selectMinimalPref(Solusions)$;
    return $Success$
  return $Failure$
End Function
```

Fig. 5. The Preference Revision Algorithm

to adjust values of attributes (that is, criteria in the AHP) of alternatives. In the above example, it may be a possible strategy for the agent (the compromiser) to increase the value of the attributes for the alternative A_2 . Namely, by adjusting the value of attribute, $C_1(A_2)$, the utility for A_2, $u(A_2) = f(f_{C_1}(C_1(A_2)), \ldots)$, will be changed. In conclusion, the preference order will be changed. In our system, a compromiser tries to adjust attribute values based on "generate and test" style. The problem is that the solution space is too huge to revise agent's preference. If there are 3 alternatives, and there are 4 attributes and each attributes has 5 possible values, then an agent must find a solution from $5^{3\times4}$ possible states. Namely, If there exist p alternatives, q attributes, and d possible attribute values, then the solution space exponentially becomes to d^{pq}. This means that an agent must take about $O(d^n)$ steps, where $n = pq$. In order to, effectively, find a solution from the solution space, we need a heuristic strategy. Therefore, we propose a preference revision strategy based on the minimal change (MC) principle and the order-based change (OC) principle. These principles can be described as follows: **The MC principle:** An agent should change an user's preference as minimal as possible. If an user's preference is change dramatically, the agent of the user will be unreliable for the user. Therefore, an agent searches a solution which minimally change an user's original preference. **The OC principle:** An agent should change an user's preference based on the preference order of alternatives. Changing the second alternative to the most preferable is easier than changing the worst alternative to the most preferable. Therefore, an agent tries to select an alternative and adjust the values of attributes of the alternative according to an user's original preference order.

Based on the above two principles, we propose the algorithm for preference revision in Figure 5. In the algorithm, the compromiser firstly tries to adjust the values of attributes for the persuader's most preferable alternative (PA). Secondly, the compromiser tries to adjust the value for his own most preferable alternative (MA). Consider that a compromiser's preference is $A_1 \succ A_2 \succ A_3$ and a persuader's most preferable alternative is A_2. In this case, firstly the compromiser tries to increase the values of attributes of A_2 in order to increase the utility of A_2. If the compromiser's most preferable alternative becomes A_2 , this preference revision succeeds. If not, the compromiser tries to decrease the values of attributes of A_1 in order to decrease the utility of A_1. Concretely, in the AHP, in order to increase/decrease a value of an attribute (criterion) for an alternative, agents increase/decrease all pairwise comparison values with respect to the alternative in pairwise comparison matrices by using the method described in our previous paper [5]. The reason why agents can adjust the pairwise comparison values is that the value is determined with fuzzy and verbal measurement by an user. The details are discussed in the paper [5].

In this algorithm, in order to determine which attributes should be adjusted, an agent creates all combination of the attributes as a power set. For example, there exist three attributes, C_1,C_2, and C_3. An agent creates the power set, $\{\{C_1\}\{C_2\}\{C_3\}\ \{C_1, C_2\}\{C_1, C_3\}\{C_2, C_3\}\ \{C_1, C_2, C_3\}\}$, where empty set is deleted. $\{C_1, C_2\}$ means that the agent tries to change the values of the attributes C_1 and C_2 . Based on the MC principle, an agent tries to adjust these attributes sets in order of the size. The function $selectMinimalPref()$ selects the best preference from the candidates based on the MC principle. In order to measure the difference between the original preference and a new candidate preference, we employ a total distance and define the following formula. $\sum_{0\leq i\leq m} |OO(A_i) - NO(A_i)|$, where $OO(A_i)$ means the original preference order of the alternative A_i, and $NO(A_i)$ means the new preference order of the alternative A_i . Consider that a compromiser's preference changed $A_1 \succ A_2 \succ A_3$ to $A_2 \succ A_1 \succ A_3$. In this case, $OO(A_1)$ is 1 and $NO(A_1)$ is 2, because the alternative A_1 is the most preferable in the original preference order and the second in the new order. Thus, $|OO(A_1) - NO(A_1)|$ is 1. Therefore, the total differences between $A_1 \succ A_2 \succ A_3$ and $A_2 \succ A_1 \succ A_3$ becomes 2 based on the above formula.

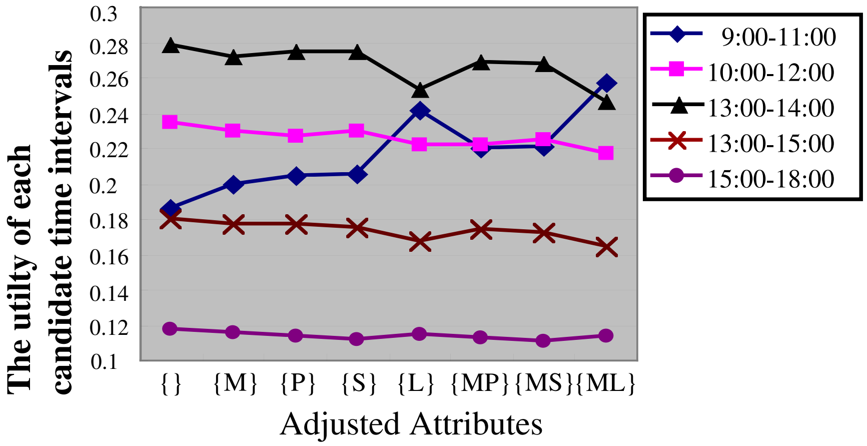


Fig. 6. Experimental result

4 Discussion

4.1 Experimental Result

Figure 6 shows the experimental result of the process of adjusting the utility of three candidate time intervals, 9:00-11:00, 10:00-12:00, 13:00-14:00, 13:00-15:00, and 15:00-18:00 by using the preference revision algorithm proposed in this paper. The vertical axis shows the utility of the alternatives. The adjusted attributes are shown in the horizontal axis. In the horizontal axis, "M" means an attribute "Members," "P" means an attribute "Place," "S" means an attribute "Size", and "L" means an attribute "Length." In this example, the agent's initial most preferable alternative is "13:00-14:00", and the agent is persuaded based on "9:00-11:00" that is the persuader's most preferable alternative. The agent tries to revise his preference by using the algorithm in Figure 5. The agent, firstly, generates a power set of the attributes, and adjusts the value of the attributes in order of the size of the set. In this example, the agent, firstly, revises the value of the attribute M; secondly he revises the value of attribute P, and so on. Finally, when he revises the value of attributes, M, and L, he can succeed revising his preference. Namely, the agent's most preferable time interval can be changed to "9:00-11:00" that is the persuader's most preferable time interval.

We have another experiment in which the members of our project additionally used a typical voting method for scheduling meetings in order to evaluate and compare our approach with the voting method. The voting method is a popular method for social decision making. The members evaluated the results of the two methods, the voting method and the distributed meeting scheduler based on the multiple negotiations,by using 5 point scale in which the number 5 is used for the best. The average of points of the voting method and our approach are

"3.22" and "3.56", respectively. The result shows that the distributed meeting scheduler can be used effectively.

4.2 Features of the Distributed Meeting Scheduler

The following (1) and (2) are the main features of the multiagent negotiation mechanism: (1) The multiple negotiations can reflect user's individual preferences. In the multiple negotiations, each agent has an opportunity to play the role of a persuader. Namely, each agent is able to persuade the others. In real society, several factors (e.g., a social position, anxiousness for evaluation from other people, and so on) prevent people from having a fair opportunity to reflect their own preferences. These factors are called a production blocking, an evaluation apprehension, and a free riding, and a cause productivity loss [1]. In our system, in order to reduce a productivity loss, agents have fair opportunity to reflecting their own preferences. Furthermore, in the multiple negotiations host-agents exchange information on processes of negotiation. By exchanging information on which member-agents accept or reject proposals, host-agents can reduce work required to persuade member-agents. Namely, exchanging information reduces times to reach a consensus amon! ! g agents. (2) The preference revision algorithm effectively narrows down the solution space. Consider that there exist p alternatives, q attributes, and d possible attribute values. The solution space becomes to d^{pq}. In this case, agents must take about $O(d^n)$ steps, where $n = pq$. The preference revision algorithm enables agents to narrow down the size of the solution space from d^{pq} to d^{2q} , since the preference revision algorithm focuses on the 2 alternatives (i.e. $p = 2$), the persuader's most preferable alternative and the compromiser's most preferable alternative. Agents must take only $O(d^q)$ steps. Since $n \geq q$, in practical applications, agents can effectively reduce their steps to find a solution.

5 Conclusions

In this paper, we proposed a new multi-agent negotiation protocol that includes the multiple negotiations and the preference revision in persuasion among agents. We have implemented a distributed meeting scheduler to see how effectively the multi-agent negotiation can be used. In the scheduling system, an agent is assigned to an user who plans private and public events. An agent negotiates with other agents about making an public schedule by referring the user's private schedules and preferences. The multiple negotiations facilitates reaching an agreement among agents effectively. The result shows that the multi-agent negotiation based on private preferences is an effective method for a distributed meeting scheduler.

References

1. M.Diehl and W.Stroebe. Productivity loss in idea-generating groups: Tracking down the blocking effect. *Journal of Personality and Social Psychology*, 61:392–403, 1991. 106
2. N.Fukuta, T.Ito, and T.Shintani. MiLog: A Mobile Agent Framework for Implementing Intelligent Information Agents with Logic Programming, In *Proc. of the 1st Pacific Rim International Workshop on Intelligent Information Agents(PRIIA'2000)*, pp.113-123, 2000. 98
3. L.Garrido and K.Sycara. Multi-agent meeting scheduling: Preliminary experiment results. In *Proc. of 2nd International Conference on Multi-Agent Systems(ICMAS-96)*, pages 95–102. AAAI Press, 1996. 96
4. T.Haynes, S.Sen, N.Arora, and R.Nadella. An automated meeting scheduling system that utilize user preferences. In *Proc. of The 1st International Conference on Autonomous Agents (Agents'97)*, pages 308–315, 1997. 96
5. T.Ito and T.Shintani. Persuasion among Agents : An approach to Implementing a Group Decision Support System based on Multi-Agent Negotiation. In *Proc. of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 592–597, 1997. 95, 100, 104
6. T. Ito and T. Shintani. On a Mechanism of Persuasion Among Agents for Group Choice Design Supprt Systems. In *Systems and Computers in Japan, Vol.29, No.5*, pages 20-28, John Wiley & Sons, Inc.,1998. 99
7. R. L. Keeney and H.Raiffa. *Decisions with Multiple Objectives : Preference and Value Tradeoffs*. Cambridge Univ. Press, 1993. 96
8. K.Kusano. *Negotiation As a Game*. Maruzen Library 130, 1994. in Japanese. 95
9. S.Parsons, C.Sierra, and N. R. Jennigs. Agents that reason and negotiate by arguing. *Journal of Logic and Computation*, 8(3):261–292, 1998. 96
10. J. S. Rosenschein and G.Zlotkin. *Rules of Encounter*. The MIT Press, 1995. 95
11. T.Saaty. *The analytic hierarchy process*. McGraw Hill, 1980. 96
12. S.Sen and E. H. Durfee. A formal study of distributed meeting scheduling. *Group Decision and Negotiation*, pages 265–289, 1998. 96
13. T.Shintani, T.Ito, and K.Sycara. Multiple negotiations among agents for a distributed meeting scheduler. In *Proc. of the 4th International Conference on Multi-Agent Systems (ICMAS-2000)*, pages 435–436, 2000. 96
14. C.Sierra, N. R. Jennings, P.Noriega, and M.Wooldridge. A framewok for argumentation-based negotiation. In M. P. Sigh, A. Rao, and M. J. Wooldridge, editors, *Proc. of the 4th International Workshop on Agent Theories, Architectures and Languages (ATAL-97)*, LNAI 1365, pages 177–192, 1997. 96
15. K. P. Sycara. Multiagent systems. *AI Magazine*, 19(2):79–92, 1998. 96
16. T.Tsuruta and T.Shintani. Scheduling meetings using distributed valued constraint satisfaction algorithm. In *Proc. of the 14th European Conference on Artificial Intelligence (ECAI-2000)*, pages 383–387, 2000.

Autoplex: Automated Discovery of Content for Virtual Databases

Jacob Berlin and Amihai Motro

Information and Software Engineering Department
George Mason University, Fairfax, VA 22030
{jberlin,ami}@gmu.edu

Abstract. Most virtual database systems are suitable for environments in which the set of member information sources is small and stable. Consequently, present virtual database systems do not scale up very well. The main reason is the complexity and cost of incorporating new information sources into the virtual database. In this paper we describe a system, called Autoplex, which uses machine learning techniques for automating the discovery of new content for virtual database systems. Autoplex assumes that several information sources have already been incorporated ("mapped") into the virtual database system by human experts (as done in standard virtual database systems). Autoplex learns the *features* of these examples. It then applies this knowledge to new candidate sources, trying to infer views that "resemble" the examples. In this paper we report initial results from the Autoplex project.

1 Introduction

The integration of information from multiple databases has been an enduring subject of research for over twenty years (for example, [16,7,10,5,21,27,26,2,11]). Indeed, while the solutions that have been advanced tended to reflect the research approaches prevailing at their time, the overall goal has remained mostly unchanged: to provide flexible and efficient access to information residing in a collection of distributed, heterogeneous and overlapping databases (more generally, other kinds of information sources may be considered as well).

A common approach to this problem has been to integrate the independent databases by means of a comprehensive *global scheme* that models the information contained in the entire collection of databases. This global scheme is fitted with a *mapping* that defines the elements of the global scheme in terms of elements of the schemes of the member databases. Algorithms are designed to interpret queries on the global scheme. Such *global queries* are translated (using the information captured in the mapping) to queries on the member databases; the individual answers are then combined to an answer to the global query. The global scheme and the scheme mapping constitute a *virtual database*; the main difference between a virtual database and a conventional database is that whereas a conventional database contains data, a virtual database points to other databases that contain the data (Figure 1). An important concern is that this

© Springer-Verlag Berlin Heidelberg 2001

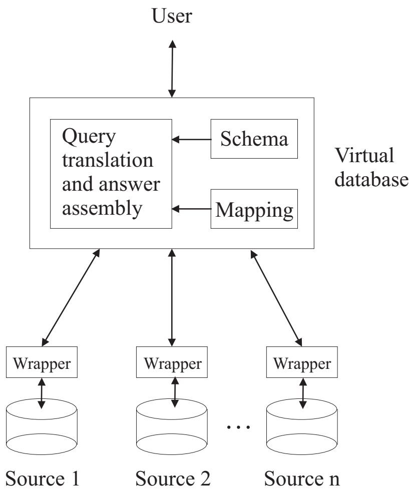


Fig. 1. Typical architecture for integrating heterogeneous information sources

query processing method be *transparent*; i.e., users need not be aware that the database they are accessing is virtual.

Although there are a number of recent systems that follow this general scheme, for example [4,13,22], none of these systems scale up to an environment in which the number of potential sources is very large, and which is constantly changing (such as the World Wide Web). The primary limitation is that the process of incorporating new member schemes into the global scheme is complex and costly. Consequently, such systems tend to be useful only when the community of member databases is small and stable. Indeed, it is commonly agreed upon in this field and the related field of data warehousing that future research should find ways to automate the integration and maintenance process [14,25].

Given the vast amount of information available and the cost of locating and incorporating such information into a virtual database, we have been developing a system, called Autoplex, for *discovering* member schemes and incorporating them into the global scheme with only limited human effort. Based primarily on Bayesian learning, the system acquires probabilistic knowledge from examples that have already been integrated into the virtual database. Our approach thus follows a supervised learning paradigm. From the acquired probabilistic knowledge, the system can discover "content contributions" in new, previously unseen information sources. Although not treated in detail in this paper, we believe this approach can also be used to maintain the contribution definitions in a dynamic environment where the underlying schemes may change over time.

The authors are unaware of any prior work that attempts to automate the discovery and mapping of new data sources. Two recent works, however, share with Autoplex the general goal of accelerating the mapping process. In [15] a neural network-based method is described, that classifies attributes of data sources. This important integration step is also part of the Autoplex discovery process.

The recent Clio system [18] introduces an interactive process that facilitates the mapping of a given source (such as a legacy database) to a target schema. However, mappings are derived from prespecified source-target relationships (called value correspondences). The translation of heterogeneous data is also the subject of [1,19]. These approaches, too, are based on prespecified correspondence rules. It must be mentioned that file translation, schema mapping, and database restructuring are ancient database problems, with seminal work, often-ignored, done over 25 years ago (for example, [17,24,23]).

The remainder of this paper is organized as follows. Section 2 states the problem formally, and provides an overview of the architecture. Section 3 discusses how probabilistic knowledge is acquired from the examples. Section 4 discusses how that knowledge is used to discover a contribution from a new source. Section 5 describes the experimental environment and initial results. Finally, Section 6 concludes with a brief discussion of proposed future work. A fuller discussion may be found in [6].

2 Basic Issues and Assumptions

In this section we describe the framework of this project, and we outline the overall architecture of Autoplex. Our work is conducted within the framework of the Multiplex virtual database system, but is of general applicability to most such systems. We begin with a brief description of Multiplex.

2.1 Multiplex

The multidatabase system Multiplex [22] is an example of a virtual database system. The basic architecture of Multiplex is fairly simple. The virtual database consists of a *global scheme* (described in the relational model) and a *mapping table*. Each entry in this table is called a *contribution* and consists of two expressions. The first expression is a view (expressed in SQL) of the global scheme; the second expression is a query to one of the member databases (expressed in the language of that system). The first expression is called a *global view*; the second expression is called a *local view*. The result obtained from the second expression is assumed to be a materialization of the view described in the first expression. The complexity of these expressions can vary greatly: they could range from a complex calculation, to a statement that simply denotes the equivalence of two attribute names.

2.2 Statement of the Problem

In its most general form, a contribution is a pair of arbitrary view expressions: one on a member scheme, the other on the global scheme. For reasons of complexity, Autoplex places limitations on these expressions. Specifically, it assumes that the global view is a single global relation, and that the local view is a *selection-projection* expression on a single relation. The challenge of Autoplex can be stated in terms of the Multiplex system as follows. *Given:*

1. A relation scheme $R = (X_1, \ldots, X_n)$. This is the virtual database. Each column X_i of R is labeled as either *required* or *optional*.
2. A set of contribution examples, each consisting of a relation scheme $S = (Y_1, \ldots, Y_k)$, a relation instance s of scheme S, and a selection-projection expression e on relation S that defines a contribution to R.
3. A new, previously unseen relation scheme $T = (Z_1, \ldots, Z_m)$ and a relation instance t of T. We shall often refer to T as a *candidate* relation.

Determine: Whether T contains an acceptable contribution to R, and if so, find the expression e_t that defines it. An acceptable contribution is one that satisfies all required columns in R and exceeds a predetermined threshold.

2.3 Autoplex Architecture

A high level overview of the Autoplex architecture is shown in Figure 2. This architecture includes two main components: a learner and a classifier.

The learner. The Bayesian learner is given the virtual relation scheme R and a set of contribution examples. Each such example consists of a local relation scheme S, an instance of this scheme s, and a selection-projection expression e on scheme S. The extension of this expression in the instance s generates tuples for the virtual relation. The Bayesian learner uses this information to acquire probabilistic knowledge on features of the examples. This knowledge is stored on secondary storage in efficient data structures for future use.

The classifier. A candidate relation scheme T and instance t are provided by a new local database as inputs to the Bayesian classifier. This classifier uses the acquired probabilistic knowledge to infer a selection-projection view that defines a contribution of T to R.

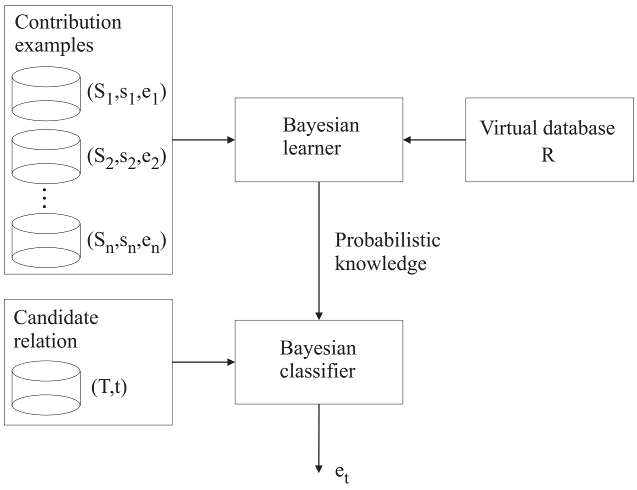


Fig. 2. Autoplex architecture

2.4 Classification Methodology

In Figure 2, classification is a single process. More precisely, classification comprises four different phases. Given a candidate relation T and a global relation R, in the first phase the classifier considers each column of T and each column of R and determines the probability that the former is an "instance" of the latter. In the second phase, the classifier finds an assignment of the local relation to the global relation that maximizes total column probabilities. At this point, if successful, the classifier has found a projection of the candidate relation that offers the best "match."

The rows of this projection now must be pruned to retain only those rows that "resemble" rows in the examples. In the third phase, the classifier partitions the instance t into two sets of rows: those that should be included in the contribution and those that should be excluded from it. Because the new contribution should be usable even after the extension of the candidate relation is updated, an *intensional* description of the included rows is desirable. In phase four, a classification tree algorithm is used to derive a selection predicate that conforms to the set of included rows.

The final output from this entire process is a selection-projection expression, or *False* in the event an acceptable contribution could not be found. Our approach is biased to search within the space of projections and selections on the candidate relation. Furthermore, our approach is greedy in that we search for a projection followed by a selection. Clearly, other approaches could produce better results. For example, a better projection may be found if some rows are first removed from the candidate. Furthermore, a better mapping may be found if we allow *transformations* of the values in the candidate (such as conversions of measurement units). Our approach represents a reasonable tradeoff between the advantage of more a powerful search (which will discover additional contributions) and the need to keep the problem tractable. More general mappings are currently under investigation (see Section 6).

3 Learning from Example Contributions

Learning from example relations is accomplished in two stages. First, we acquire probabilistic knowledge on the "behavior" of columns (to be used in determining the projective transformation of a candidate relation). Next, we acquire probabilistic knowledge on the "behavior" of rows (to be used in determining the optimal selective transformation of a candidate relation). In this section, we address both of these procedures in order. We begin with a brief survey of the Bayesian concepts that will be used throughout this work.

3.1 Bayesian Framework

Our discussion is mostly conventional in the application of Bayes Theorem to machine learning. In machine learning terminology, the problem of associating a new instance with one of previously learned classes is called *classification*. Let $P(c)$ be the *prior* probability that classification c holds before observing any data, let $P(D)$ represent the unconditional probability of observing data D, and let $P(D|c)$ represent the conditional probability of observing the data D given that the classification c holds. Bayes Theorem states:

$$P(c|D) = \frac{P(D|c)P(c)}{P(D)} \ . \tag{1}$$

$P(c|D)$ is referred to as the *posterior* probability of c, because it reflects the probability of classification c after the data D has been observed. In machine learning problems, we wish to find the best classification. Translated into Bayesian terms, we wish to find the classification with the greatest posterior probability given the data. Since the probability of the data $P(D)$ is common for all classifications, the most probable classification is the one that maximizes the numerator in Equation 1. Letting D be a sequence of attribute values $(d_1, d_2, \ldots, d_n)$ and C a set of possible classifications $(c_1, c_2, \ldots, c_m)$, we wish to determine the classification $c_i \in C$ such that $\forall c_j \in C$ and $c_j \neq c_i$,

$$P(d_1, d_2, \ldots, d_n|c_i)P(c_i) \geq P(d_1, d_2, \ldots, d_n|c_j)P(c_j) \ . \tag{2}$$

The terms in Equation 2 do not produce probabilities since we have removed the denominator; however, the terms can be easily converted to probabilities by normalization. For our purposes, we shall make the *naive* assumption that data values are conditionally independent given the classification. This assumption is the basis for the well-known Naive Bayes classifier. Both theoretical and experimental results show that even when the independence assumption does not hold, the Naive Bayes classifier performs comparable to or better than other more sophisticated approaches in many problem domains [20,12,8]. The key to this paradox is that good classifications can be made from this approach even when the estimated probabilities are wrong. From the independence assumption, we can rewrite Equation 2 as

$$P(c_i) \prod_{k=1}^{n} P(d_k|c_i) \geq P(c_j) \prod_{k=1}^{n} P(d_k|c_j) \ . \tag{3}$$

Equation 3 is useful in our problem domain for several reasons. First, since we assumed conditional independence, the number of probability terms that we need to estimate has been reduced to a manageable level. Second, the equation is robust to noise since observed data only causes incremental changes to our estimates. Finally, the equation is robust to missing features since we simply ignore the conditional probability terms for the missing features. All of these aspects of Equation 3 make a Bayesian approach towards automated discovery of contributions attractive.

3.2 Learning Column Behavior

Our strategy is to learn the "behavior" of each column X of R and, upon encountering a new column Z, determine the posterior probability that the new column is in the class of X, given the data values of Z. For this, we use a collection of Naive Bayes (NB) classifiers: one classifier for each column X. Building a collection of classifiers instead of just one gives us the ability to determine that a candidate column should *not* be mapped to any of the columns of R or that it may be mapped to *more* than one column of R.

To construct a classifier for column X we need to learn the prior probability $P(X)$ that an arbitrary column maps to X, and, for each column, the conditional probability of that column among the columns that were mapped to X. Following our assumption of conditional independence among the values of a column, the latter is substituted by the conditional probability $P(v|X)$ that an *individual* column value v occurs in the set comprising the values in columns that were mapped to X. Together, these will allow us to calculate the right hand side of Equation 3 and subsequently the best classification for the column Z. The learning algorithms are discussed next.

Learning Prior Probabilities $P(X)$, the probability that an arbitrary column maps to X, is estimated by the proportion of columns in the entire set of examples that have been mapped to X.

Learning Conditional Probabilities In estimating conditional probabilities we distinguish between values that are *words* and values that are *numbers*. The algorithm for learning conditional probabilities for words is shown in Table 1. This algorithm is similar in spirit to the typical Naive Bayes approach for document classification discussed in [20]. For a text column X, the positive examples come from all the text columns of examples that are mapped to X. The negative examples for the same X come from all the text columns of examples that are not mapped to X.

Table 1. Algorithm for learning column conditional probabilities

```
$V \leftarrow$ All distinct words in all the example relations $S$
For each text column $X$ in $R$:
    Initialize bag variables $P$ and $N$ to $\emptyset$.
    For each text column $Y$ in all example relations $S$:
        If $Y$ is mapped to $X$
            $P \leftarrow P \cup \{$All words in $Y\}$
        else
            $N \leftarrow N \cup \{$All words in $Y\}$
    For each word value $v$ in $V$:
        $C_P \leftarrow$ Number of times $v$ appears in $P$
        $C_N \leftarrow$ Number of times $v$ appears in $N$
        $P(v|X) \leftarrow \frac{C_P+1}{|P|+|V|}$
        $P(v|\neg X) \leftarrow \frac{C_N+1}{|N|+|V|}$
```

$P(v|X)$, the probability that a vocabulary word v occurs in a column mapped to X, is estimated by the proportion of the occurrences of v among words from columns that are mapped to X. This ratio is adjusted with the m-estimate to prevent zero probability terms which would dominate all calculations. We assume uniform priors for the words and weight them by the size of the vocabulary.

To estimate the probabilities of numeric values, we assume that these values conform to some distribution function; currently, a normal distribution is assumed. The learning phase consists simply of calculating the mean and standard deviation for the positive and negative examples. For a numeric column X, the positive examples come from all the numeric columns of examples that are mapped to X. The negative examples for the same X come from all the numeric columns of examples that are not mapped to X. Given a numeric value from a candidate column, we can now calculate the conditional probabilities by using the probability density function for the normal distribution.

3.3 Learning Row Behavior

Our learning process is similar to that for column behavior, except that now our evidence is rows of data instead of columns of data. In the learning phase we shall examine the rows of each example relation S (noting which rows were selected and which were discarded). Upon encountering a new table (whose columns have already been matched successfully to the columns of R) we shall use our acquired knowledge to determine, for each row, whether it should be selected or not.

To determine whether a row of a future table should contribute to R, we need to learn the prior probability $P(R)$ that an arbitrary row is a contributor to R, and, for each row (a *sequence* of values), the conditional probability of that row within the set of selected rows. Again, following our assumption of conditional independence among the components of a row, we learn instead the conditional probabilities $P(v|R)$ that an *individual* row component v occurs in the set comprising the values taken from the same column position in all the selected rows. Together, these will allow us to calculate the right hand side of Equation 3 and subsequently the appropriate classification for each row.

Learning Prior Probabilities $P(R)$, the probability that an arbitrary row would contribute to R, is estimated by the proportion of rows in the entire set of examples that were selected.

Learning Conditional Probabilities For textual columns, we use the algorithm in Table 2 to learn the conditional probabilities of words. This algorithm is similar to the one in Table 1, except that now we process rows of data. For numeric columns, we again assume a normal distribution on the values and calculate the mean and standard deviation for the examples. For both textual and numerical values, positive examples are drawn from the set of rows that satisfy the selection predicate of the contribution. Negative examples are drawn from the set of rows that do not satisfy the selection predicate.

Table 2. Algorithm for learning row conditional probabilities

```
For each text column $X$ of $R$:
    $V \leftarrow$ All distinct words from all example columns mapped to $X$
    Initialize bag variables $P$ and $N$ to $\emptyset$.
    For each example $S$ and its selection predicate $\sigma$:
        For each row $t$ from $s$:
            For each column $Y$ of $S$ that was mapped to $X$:
                If $t$ satisfies $\sigma$
                    $P \leftarrow P \cup t[Y]$
                else
                    $N \leftarrow N \cup t[Y]$
    For each word value $v$ in $V$:
        $C_P \leftarrow$ Number of times $v$ appears in $P$.
        $C_N \leftarrow$ Number of times $v$ appears in $N$.
        $P(v|R) \leftarrow \frac{C_P+1}{|P|+|V|}$
        $P(v|\neg R) \leftarrow \frac{C_N+1}{|N|+|V|}$
```

4 Discovering New Contributions

Given a candidate relation, discovery is accomplished in two phases. First we search for a projective transformation of the candidate relation to match the global relation R. If successful, we then search for a selective transformation to be applied subsequently. We shall discuss each of these procedures in order.

4.1 Projective Transformations

Discovering a projective transformation of a candidate relation is a two-step process. For each column of the candidate relation and for each column of the virtual relation we use a Naive Bayes (NB) classifier to estimate the probability that the columns match. This information is represented in a bipartite weighted graph, and a graph algorithm is applied to find the optimal overall matching of the candidate relation to the virtual relation. This optimal mapping is our projective transformation.

The process for estimating probabilities is shown in Table 3. Because we assume conditional independence among the values (even when this assumption is inaccurate), the outcome of this algorithm may not be strictly the probability of a match; hence, we use the notation $\hat{P}$.

The output of the NB mapping algorithm is a bipartite graph in which the columns of T and R are represented by nodes in two partitions, each column mapping is represented by an edge and the mapping probabilities are edge weights. The simplest way to approach optimality is to look for a maximum weighted matching in this bipartite graph [9]. After removing "weak" edges that fall below a user specified threshold, we search the graph for a subset of the edges that

Table 3. Estimating the probabilities for columns of T matching columns of R

```
For each column $X$ in $R$:
    For each column $Z$ in a candidate relation $T$:
        Let $v_1, \ldots, v_n$ be the values of $Z$.
        $\hat{P}(match) = P(X) \prod_i P(v_i|X)$
        $\hat{P}(\neg match) = P(\neg X) \prod_i P(v_i|\neg X)$
        Normalize $\hat{P}(match)$ and $\hat{P}(\neg match)$ to sum to 1.
        Output $\hat{P}(match)$ as probability that $Z$ maps to $X$.
```

connects nodes in one partition to the nodes in the other partition such that no two edges share a node and the sum of the weights is maximal. The classifier uses a polynomial-time algorithm [3] to find the maximum weighted matching. This matching is then translated into a relational algebra expression that defines the projective contribution of T to R.

4.2 Selective Transformations

Discovering a selective transformation of a candidate relation is also a two-step process. First, we partition the rows of the candidate into a set of contributing rows and a set of non-contributing rows, and we label the rows accordingly. Next, we apply a classification tree algorithm to the labeled rows, to learn a set of rules that *defines* the partition; these rules are then converted to a selection predicate. A more detailed discussion follows.

Labeling Rows of a Candidate Table Given a candidate relation, we use the acquired probabilistic knowledge to partition its set of rows to those that should be selected and those that should be discarded. We use the algorithm in Table 4 to estimate the probability that a candidate row should be included as a contribution to R. Rows that have a probability greater than a user-specified threshold are labeled as members of the contributing set; the remaining rows are labeled as members of the non-contributing set.

Table 4. Estimating the probabilities for rows of T contributing to R

```
For each row $t$ in a candidate relation $T$:
    $\hat{P}(R|t) = P(R) \prod_i P(v_i|R)$
    $\hat{P}(\neg R|t) = P(\neg R) \prod_i P(v_i|\neg R)$
    Normalize $\hat{P}(R|t)$ and $\hat{P}(\neg R|t)$ to sum to 1.
    Output $\hat{P}(R|t)$ as probability that row $t$ contributes to $R$.
```

Inferring Selection Predicates We use a standard classification tree algorithm (J48 from the WEKA machine learning package [28]) to find a selection predicate that defines the contributing set of rows. Before applying the candidate rows to the classification tree learner, some preprocessing is necessary to initialize the learning algorithm with the column names, column types, and the set of values for each column.

The learned classification tree represents a disjunction of conjunctive rules on the column values. These rules partition the data according to their labels. Since the classification tree considers all of the columns of the candidate, it is likely that the tree will use the unmapped columns of the candidate in the set of rules. This is interesting because it allows us to define selection predicates on columns that are neither in the virtual database nor in the example contributions.

For our problem, we are interested in the rules that define the contributing rows. Through simple string parsing, we build a selection predicate as a disjunction of the conjunctive rules that identify contributing rows.

5 Experimentation

We built a prototype in the Java programming language to test the ideas in this paper. This prototype is not yet integrated with a complete virtual database system, such as Multiplex. Specifically, the experimental data was collected off-line from the World Wide Web and stored locally in relational database tables.

5.1 Measures of Performance

To measure performance, the outputs of Autoplex are regarded as four types of Boolean decisions:

1. *Column Mapping:* For each combination of a candidate column and a virtual column, decide whether or not the columns match.
2. *Table Mapping:* For each combination of a candidate table and a virtual table, decide whether or not the tables match.
3. *Tuple Partitioning:* For each tuple in a candidate table, decide whether to assign it to the contributing set or to the non-contributing set.
4. *Tuple Selection:* After we infer a selection predicate from the partitioned tuples, decide for each tuple whether or not it satisfies the selection predicate.

Obviously, the last two decisions are made only if we have made a positive table mapping decision. For each type of decision, the output falls into four disjoint categories:

A. Decision is *True* and the correct answer is *True* (*true positives*).
B. Decision is *False* and the correct answer is *True* (*false negatives*).
C. Decision is *True* and the correct answer is *False* (*false positives*).
D. Decision is *False* and the correct answer is *False* (*true negatives*).

The ratio $|A|/(|A|+|C|)$ is the proportion of true positives among the cases thought to be positive; i.e. it measures the accuracy of Autoplex when it decides *True*. The ratio $|A|/(|A|+|B|)$ is the proportion of positives detected by Autoplex among the complete set of positives; i.e. it measures the ability to detect positives. Specifically to our application, the former ratio measures the *soundness* of the content that has been discovered, and the latter ratio measures the *completeness* of the discovery process. These two ratios are known from the field of information retrieval as *precision* and *recall*, but we shall refer to them here as the soundness and completeness of the discovery process. Thus, we can measure the soundness and completeness of column mapping, table mapping, tuple partitioning, and tuple selection.

5.2 Setting Up the Experiment

To experiment with the prototype, we defined a virtual database for computer retail information with the following relations:

1. Desktops = (*Retailer, Manufacturer, Model*, Cost, Availability)
2. Monitors = (*Retailer, Manufacturer, Model*, Cost, Availability)
3. Printers = (*Retailer, Manufacturer, Model*, Cost, Availability)

Italicized attributes denote primary keys. The *Retailer* attribute is derived from the information source (for example, we use the web address). For the purposes of our experiment, the primary keys are *required* for a candidate to be accepted as a contribution to the virtual database. All other fields are *optional*.

Data for this experiment was taken from the web sites of 15 different computer retailers (e.g. Gateway, Egghead, etc). The data was collected off-line from HTML web pages and imported into relational database tables accessible through the ODBC protocol. The data from each retailer was imported into a single local table and then mapped to one or more virtual tables through selective and projective transformations.

To experiment with this data, we used a procedure from data mining called *stratified threefold cross-validation* [28], which we briefly describe. Each of the 15 web sources was manually mapped into our virtual database by using a mapping table as discussed in Section 2.1. We partitioned the 21 mappings in our mapping table into three folds of approximately equal content. Using two folds for learning and one fold for testing, we repeated the experiment for the three possible combinations of folds. To measure the soundness and completeness of the discoveries, the information in the mapping table was assumed to be the correct mapping of these sources.

5.3 Results

Table 5 shows the soundness and completeness for the four types of decisions made by Autoplex. A perfect sequence of decisions would result in $Soundness = Completeness = 1$. It is interesting to note that soundness and completeness for column mapping are approximately equal. This behavior is caused by the matching constraint of the weighted bipartite graph algorithm (no two edges may share a node). Due to this constraint, an incorrect decision to map two columns typically causes both a false positive and a false negative.

Table 5. Soundness and completeness for column mapping decisions

Category	A	B	C	D	Soundness	Completeness
Column Mapping	74	17	17	660	0.81	0.81
Table Mapping	18	3	0	24	1.00	0.86
Tuple Partitioning	969	60	117	612	0.89	0.94
Tuple Selection	967	62	104	625	0.90	0.94

Soundness and completeness for table mapping are higher than column mapping due to our constraint that all required columns of the virtual table must be mapped to make a positive table mapping decision. Given a candidate table and a virtual table that should not be mapped, we must determine that only one of the required virtual columns should not be mapped to any of the columns in the candidate to prevent a false positive. Thus, false positives in column mapping do not entail false positives in table mapping. Furthermore, optional columns that are not mapped do not cause a false negative for table mapping.

Table 5 also shows the performance for tuple selection is slightly better than the performance for tuple partitioning. This improvement is due to the pruning strategy employed by the J48 classification tree algorithm that learns selection predicates. Pruning prevents the overfitting of noise in partitioned sets of tuples. This strategy works well in our experiment because we have errors in the partitioned sets of tuples that are used for classification tree learning.

6 Conclusion

In this paper we described a novel approach to virtual databases, aimed at solving a serious limitation of all such systems: the cost and complexity of incorporating new data sources into the global system. This limitation hampers scalability, in effect restricting the virtual database paradigm to applications in which the community of sources is relatively small and stable.

The results of our initial experimentation are encouraging enough to support our main thesis of automatic discovery of content for virtual databases. Among the many research issues that are on our agenda, we discuss briefly four issues.

Support more general views. The ideal virtual database system can map a local source to a global database by matching arbitrary views of the local scheme with arbitrary views of the global scheme. In the version of Autoplex we described, the local view is a selection-projection of a single relation, and the global view is simply a relation. Our first priority is to support more general views, and we mention here two examples: (1) Allow local and global views that involve *joins*; i.e., discover content in a join of two *local* relations, and discover content for a join of two *virtual* relations. (2) Discover content that becomes suitable for a virtual database after an appropriate *transformation*; e.g., a local column would be mapped if it matches a virtual column after a linear transformation (such as the conversion of Fahrenheit to Celsius).

Use intensional information. The features considered in this paper were purely extensional. Yet, intensional information, such as *integrity constraints* on the virtual database, could be used to improve the discovery process. Roughly speaking, with extensional features, discoveries are based on "similarity" to example data. With intensional features, discoveries would also be based on the satisfaction of constraints. We are confident that the future incorporation of intensional features will improve the performance of Autoplex.

Assurance Contributions in Autoplex are adopted with different levels of assurance in their suitability. These levels of assurance should be remembered so that answers to database queries could be annotated accordingly. The initial work in this research focused on the statistical performance of the discovery process using standard cross-validation techniques. This work will be extended to combine the statistical performance of the discovery process with confidence measures of individual discoveries to produce a meaningful assurance measure.

Application to data warehouses. The premise of virtual databases is closely related to that of *data warehouses*, in that both create global repositories that integrate data from multiple, heterogeneous data sources. The techniques described in this paper have the potential for developing a type of data warehouse, which (after a preliminary phase of design and learning) is populated and maintained automatically by "crawlers" that periodically visit the data sources.

References

1. S. Abiteboul, S. Cluet, and T. Milo. Correspondence and translation for heterogeneous data. *Proc ICDT*, pages 351–363, 1997. 110
2. R. Ahmed, P. De Smedt, W. Du, W. Kent, M. A. Ketabchi, W. A. Litwin, A. Rafii, and M. C. Shan The Pegasus heterogeneous multidatabase system. *IEEE Computer*, 24(12):19–27, 1991. 108
3. Algorithmic Solutions. *The LEDA Users Manual (Version 4.2.1)*, 2001. 117
4. Y. Arens. Query Reformulation for dynamic information integration. *Journal of Intelligent Information Systems*, 6:99–130, 1996. 109
5. C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *Computing Surveys*, 18(4):323–364, Dec 1986. 108
6. J. Berlin and A. Motro. Autoplex: Automated discovery of content for virtual databases. Technical Report ISE-TR-00-04, George Mason University, August 2000. 110
7. U. Dayal and H. Hwang. View definition and generalization for database integration in a mutlidatabase system. *IEEE ToSE*, SE-10(6):628–644, November 1984. 108
8. P. Domingos and M. Pazzani. Conditions for the optimality of the simple Bayesian classifier. *Proc ICML*, pages 105–112, 1996. 113

9. Z. Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys*, 18(1):23–38, March 1986. 116
10. D. Heimbigner and D. McLeod. A federated architecture for information management. *ACM ToIS*, 3(3):253–278, July 1985. 108
11. Y. Kambayashi, M. Rusinkiewicz, and A. Sheth, editors. *Proc RIDE-IMS*, 1991. 108
12. P. Langley, W. Iba, and K. Thompson. An Analysis of Bayesian Classifiers. *Proc of the Tenth National Conference on AI*, pages 223–228, 1992. 113
13. A. Levy. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems*, 5(2):121–143, September 1995. 109
14. A. Levy, C. Knoblock, S. Minton, and W. Cohen. Information integration. *IEEE Intelligent Systems*, 13(5):12–24, 1998. 109
15. W-S. Li and C. Clifton. Semantic integration in heterogeneous databases using neural networks. In *Proc VLDB*, pages 1–12, 1994. 109
16. W. Litwin. MALPHA: A relational multidatabase manipulation language. In *Proc ICDE*, pages 86–93, 1984, 108
17. A. G. Merten and J. P. Fry. A data description language approach to file translation. In *Proc of ACM-SIGFIDET*, 1974. 110
18. R. Miller, L. Haas, and M. Hernández. Schema mapping as query discovery. *Proc VLDB*, pages 77–88, 2000. 110
19. T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. *Proc VLDB*, pages 122–133, 1998. 110
20. T. Mitchell. *Machine Learning*. McGraw-Hill, 1997. 113, 114
21. A. Motro. Superviews: Virtual integration of multiple databases. *IEEE Transactions on Software Engineering*, SE–13(7):785–798, July 1987. 108
22. A. Motro. Multiplex: a formal model for multidatabases and its implementation. *Proc NGITS*, pages 138–158, 1999. 109, 110
23. S. B. Navathe and J. P. Fry. Restructuring for large databases: three levels of abstraction. *ACM ToDS*, 1(2), June 1976. 110
24. J. A. Ramirez, N. A. Rin, and N. S. Prywes. Automatic generation of data conversion programs using a data description language. In *Proc ACM-SIGFIDET*, 1974. 110
25. E. A. Rundensteiner, A. Koeller, and X. Zhang. Maintaining data warehouses over changing information sources. *Communications of the ACM*, 43(6):57–62, 2000. 109
26. P. Scheuermann, C. Yu, A. Elmagarmid, H. Garcia-Molina, F. Manola, D. McLeod, A. Rosenthal, and M. Templeton. Report on the workshop on heterogeneous database systems. *SIGMOD Record*, 19(4):23–31, December 1990. 108
27. A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous and autonomous databases. *Computing Surveys*, 22(3):183–236, Sep 1990. 108
28. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 2000. 118, 119

Cooperation Strategies for Information Integration

Maurizio Panti, Luca Spalazzi, and Loris Penserini

Istituto di Informatica, University of Ancona,
via Brecce Bianche, 60131 Ancona, Italy
{panti,spalazzi,pense}@inform.unian.it

Abstract. We discuss and analyse cooperation strategies for rewriting queries in a mediator architecture. According to this approach, the mediated schema of each mediator is dynamically updated through the cooperation with information sources and other mediators, strongly influenced by the queries submitted by a consumer. This approach allows a mediator to face systems where information sources and consumer needs are dynamic. From the analysis of different cooperation strategies arises that it is more efficient and effective to directly cooperate with information sources when the sources are few. Otherwise, it is more efficient to cooperate with other mediators.

1 Introduction

Nowadays, information systems can be thought as collections of *information sources* and *information consumers* that are often *distributed* in world-wide networks. This means that sources and consumers can be autonomous and, thus, are often *heterogenous* and *dynamic*. Autonomous information sources are dynamic, since they may be added to the system, may become (temporarily or definitively) unavailable, or, sometimes, may also vary their conceptual schemas. Information consumers are dynamic as well. Indeed new consumers can be inserted or removed. Furthermore, their information needs can change very often. In such a context, information integration consists in constructing answers to queries from consumers. The most significative approach to information integration relies on the so called *mediator architecture* [10]. In this architecture, the so called mediator deals with the reformulation of a given query in a set of queries, each targeted at a selected source. Intuitively, such a reformulation should be equivalent to the original query (i.e., it should denote the same set of instances). Nevertheless, often this is not possible. Therefore, a query reformulation *contained in* (instead of *equivalent to*) the original query is considered adequate (see for example [9,2]). Furthermore, when a *description logic* [3] is used as data modeling and query language (e.g. [2,6,9]), the problem of query containment corresponds to the subsumption problem. In the *declarative* approach to intelligent information integration (e.g., SIMS [1], Information Manifold [6], and Infomaster [5]), mediators use suitable mechanisms to rewrite queries according to source descriptions (views). When consumers' needs or information sources

© Springer-Verlag Berlin Heidelberg 2001

change, new source descriptions are needed. Few systems allow automatic adaptation of such source descriptions (e.g. [7,8]), i.e., they have dynamic mediated schemas instead of static ones. In such systems, it becomes crucial the choice of right cooperation strategies in order to update their mediated schemas. Usually, every time a source changes, it communicates its change to mediators (e.g., [8]). This approach does not scale up in situations with a lot of autonomous sources (e.g. the web). Indeed, mediators and the network are overloaded with updating operations.

In this paper, we focus on cooperation strategies of mediators. Indeed, we compare different strategies from a theoretical point of view and discuss the results. The paper is organized as follows. An overview of the present work is provided in Section 2. Section 3 introduces the possible reasons of failures and the related strategies to adopt. Section 4 describes some criteria to choose partners to cooperate with. Section 5 deals with the issue of choosing the queries to send to partners. Finally, Section 6 deals with answers to ask for. Some conclusions are given in Section 7.

2 System Overview

Our goal is to build an information system capable of interconnecting information consumers and sources. Previous approaches solve several problems concerned with distributed, heterogeneous information systems. Their main limitation is related to the capability of evolving according to dynamic information systems. We propose (see also [7]) a multi-agent system which is based on a mediator architecture, i.e. mediators, sources, and consumers are agents (no matter where they are physically located). All the agents adopt a description logic called C–Classic [3] as data modelling and query language. Indeed, subsumption in C–Classic can be computed in polynomial time.

2.1 Information Source

In our system, information source agents must be able to rewrite queries depending on their local schemas. Therefore, each source must be able to *decompose* an input query into basic components and to find views over its schema that are *subsumed* by query components. Notice that, usually, information sources do not have such capabilities; we overcome this fact with appropriate wrappers that are able to acquire data and schemas from sources, to provide decomposition and classification services, and to interface sources with the rest of the system.

Example 1. Let us consider an information system which contains data about computer science articles and their authors. This information system is composed by several sources, consumers, and mediators. Figure 1 depicts three information sources of the system with their own conceptual schemas. Let us suppose that source w_1 receives a query $Q_0 \doteq \forall type.\{``Trans''\}$. w_1 finds that Q_0 subsumes *acm_trans* and *ieee_trans* and thus returns the corresponding instances (denoted by $I_{w_1}(acm\_trans) \cup I_{w_1}(ieee\_trans)$).

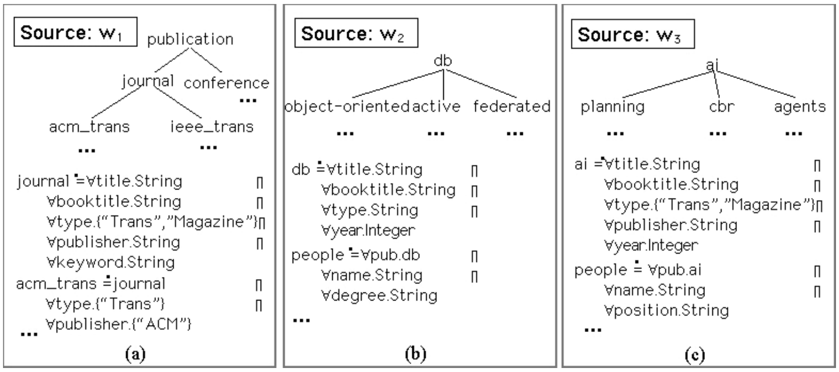


Fig. 1. An example of information sources

2.2 Mediator

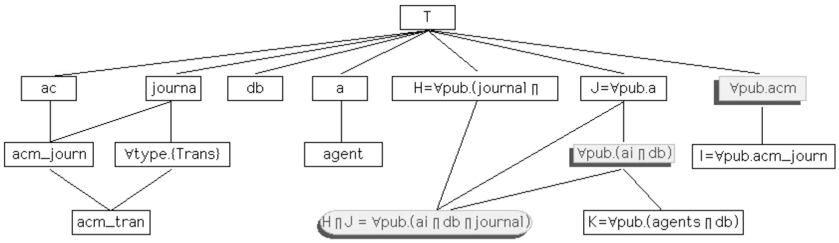
The capability of a mediator agent to face heterogeneous and dynamic systems relies on a thesaurus and a mediated schema.

Thesaurus. Each mediator has its own thesaurus in order to solve *name heterogeneity.* A thesaurus is composed by a set of classes of synonyms. Each element of a class has the reference to information sources that use it as term. Every time a new query arrives, its terms are translated in standard terms by means of the thesaurus. The reverse process occurs when the rewritten query must be sent to information sources. The thesaurus must be dynamically updated when a change in the system occurs. The classes of the thesaurus are updated by means of clustering. The explanation of this technique is out of the scope of the paper, for more details we remind to [4]. Therefore, in the rest of the paper we assume that *no name heterogeneity occurs* (i.e., the thesaurus has a maximum precision).

Mediated Schema. Each mediator has also its own mediated schema in order to solve *heterogeneity of schemas and consumer's needs.* A mediated schema is composed by a collection of **views** (that we call cases, as well). In our system [7], a view contains a consumer's query and its reformulation both represented in C–Classic. The reformulation must be subsumed by the query according to the *query containment assumption* [2,9]. Therefore, a view is a structure as the following: $\langle Q, Sol(Q), \langle (Q_1, w_1), \ldots (Q_n, w_n) \rangle \rangle$ where $w_1, \ldots, w_n$ are information sources (or other mediators that act as sources for this mediator) and $Q_1, \ldots, Q_n$ are queries to $w_1, \ldots, w_n$ respectively. Q is a given query and $Sol(Q)$ is its reformulation, i.e., a set of arbitrary combinations of $Q_1, \ldots, Q_n$ such that each element of $Sol(Q)$ is subsumed by Q. A **terminology** (i.e., a set of concept definitions[1] and their relations) is always associated to a mediated schema.

Local Query Rewriting. Every time a new query arrives, it is *decomposed* in basic components, each component is inserted in the terminology, and is *classified*

[1] $\top, \bot$ represent the *top* and *bottom* concepts, respectively.



$$\begin{array}{l}\{\ldots journal \dot{\leq} \top, db \dot{\leq} \top, ai \dot{\leq} \top, agents \dot{\leq} ai, acm \doteq \forall publisher.\{\text{“}ACM\text{”}\},\\ \quad acm\_journal \doteq journal \sqcap acm, H \doteq \forall pub.(journal \sqcap db),\\ \quad I \doteq \forall pub.acm\_journal, J \doteq \forall pub.ai, K \doteq \forall pub.(agents \sqcap db) \ldots\}\end{array}$$

Fig. 2. An example of terminology

by means of subsumption [7]. The views that are subsumed by the new query can be used as solution for the new problem. The closest to the new query the retrieved views are, the best the solutions are. This drives us to the notion of views maximally contained in the new query, [2,9]. When the mediator has a reformulation of the received query, it must send such a reformulation to related information sources and wait for their answers.

Example 2. Let us consider a mediator that has the terminology of Fig. 2 and the following views in its mediated schema:

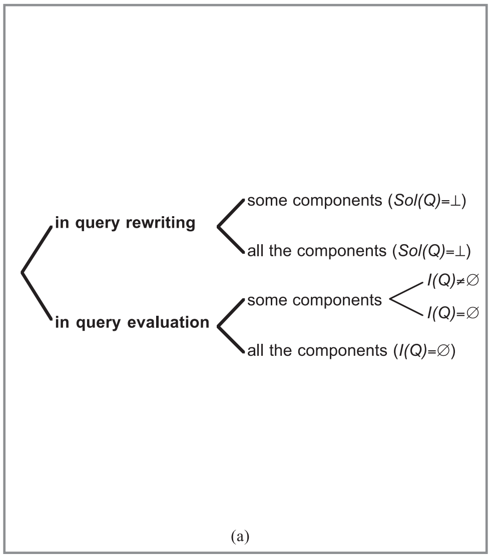
$$\begin{array}{lll}\langle H, & \langle(\forall pub.journal \sqcap \forall pub.db), (\forall pub.ai \sqcap \forall pub.db)\rangle, & \\ & \langle(\forall pub.journal, w_1), (\forall pub.db, w_2), (\forall pub.ai, w_3)\rangle & \rangle \\ \langle I, & \langle(\forall pub.acm\_trans), (\forall pub.\forall type.\{\text{“}Trans''\} \sqcap \forall pub.\forall publisher.\{\text{“}ACM''\})\rangle, & \\ & \langle(\forall pub.acm\_trans, w_1), (\forall pub.\forall type.\{\text{“}Trans''\}, w_3), & \\ & \;(\forall pub.\forall publisher.\{\text{“}ACM''\}, w_3)\rangle & \rangle \\ \langle J, & \langle(\forall pub.\forall keyword\{\text{“}AI''\}), (\forall pub.ai)\rangle, & \\ & \langle(\forall pub.\forall keyword\{\text{“}AI''\}, w_1), (\forall pub.ai, w_3)\rangle & \rangle \\ \langle K, & \langle(\forall pub.\forall keyword.\{\text{“}Agents\text{”}\} \sqcap \forall pub.db), (\forall pub.agents \sqcap \forall pub.db)\rangle, & \\ & \langle(\forall pub.\forall keyword.\{\text{“}Agents\text{”}\}, w_1), (\forall pub.db, w_2), (\forall pub.agents, w_3)\rangle & \rangle \end{array}$$

Let us suppose to have $Q_1 \doteq \forall pub.(ai \sqcap db) \sqcap \forall pub.acm$ as input query. The basic components are $Q_{1,1} = \forall pub.(ai \sqcap db)$ and $Q_{1,2} = \forall pub.acm$. Classifying such components, we obtain that $Q_{1,1}$ subsumes the views K and $H \sqcap J$, whereas $Q_{1,2}$ subsumes the view I (see Fig. 2). Therefore, the rewriting of Q_1 is the following set of queries:

$$(Sol(K) \sqcap Sol(I)) \cup (Sol(H) \sqcap Sol(J) \sqcap \; Sol(I))^{2}$$

At this point, the mediator can send the reformulation to the appropriate sources.

2 $Sol(H) \sqcap Sol(J)$ means that each element of $Sol(H)$ is conjuncted with each element of $Sol(J)$.


in query rewriting
some components (Sol(Q)=⊥)
all the components (Sol(Q)=⊥)
in query evaluation
some components
I(Q)≠∅
I(Q)=∅
all the components (I(Q)=∅)
(a)

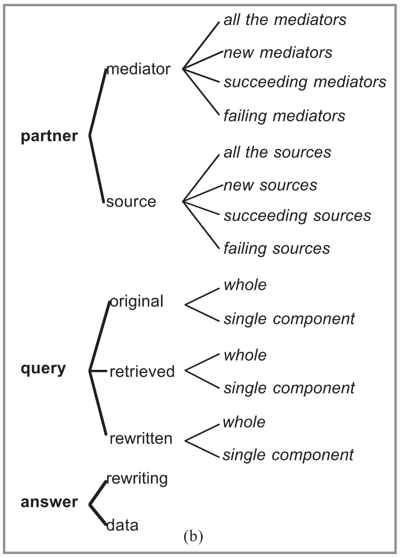


Fig. 3. Taxonomies of failures (a) and cooperation strategies (b)

3 Failures and Strategies

Failures. We have a *local failure in query rewriting* when a mediator is not able to rewrite some or all the components of a given query Q, i.e., $Sol(Q) = \perp$[3] (see Fig. 3.a). This means that the mediator's schema contains no views that can be used to reformulate the components of Q. Usually, this means that it is the first time that the consumer formulates such a query; in other words, the consumer has a new information need.

Example 3. Let us consider the mediated schema of Example 2 and the query $Q_2 \doteq \forall pub.(ai \sqcap db) \sqcap \forall affiliation.\{\text{“}Stanford\text{”}\}$. The mediator is not able to rewrite the query. Indeed, we have

$$(Sol(K) \sqcap \perp) \cup (Sol(H) \sqcap Sol(J) \sqcap \perp) = \{\perp\}$$

We have a *local failure in query evaluation* when a mediator sends a rewritten query to related sources and receives at least an empty answer. This may be enough to make empty the overall answer (see Fig. 3.a). This kind of failure means that the mediator's schema is not updated. Typically, an information source has been removed from the system or changed its schema.

Example 4. Let us suppose that source w_1 has been (perhaps temporarily) removed, then the evaluation of part of the reformulated query in Example 2 fails. Indeed $I_{w_1}(\forall pub.journal) = I_{w_1}(\forall pub.acm\_trans) = I_{w_1}(\forall pub.\forall keyword.\{\text{“}AI\text{”}\})$

[3] Notice that $I(\perp) = \emptyset$.

$= I_{w_1}(\forall pub.\forall keyword.\{“Agents”\}) = \emptyset$. Notice that the overall answer is not empty (i.e., $I(Q_1) \neq \emptyset$); indeed, the reformulations involving other sources are still valid[4].

Whatever failure occurs, the failing view in the mediator's memory must be made unavailable for a later use.

Cooperation Strategies. Even if a local failure occurs, the system has still a possibility of solving the problem by means of cooperation with other mediators and sources (*distributed query rewriting*). Notice that, this provides the system the most important way of learning. Indeed, if the query is reformulated, the new rewritten query and/or the new sources can be stored as a new view. In such a way, the mediator can support dynamic information systems. Furthermore, it does not need to maintain consistent its mediated schema every time something changes, but only when a consumer sends a query that fails, i.e., when a previous change affects the current query result. Therefore, the distributed information system (and the network, as well) is not overloaded with consistency maintenance operations, that usually are time consuming. Cooperation strategies can be classified according to three coordinates: partners, queries, and answers (see Fig. 3.b). In the rest of the paper, we describe such strategies, we also analyse them (from a theoretical point of view), since the choice of the right cooperation strategy is crucial for effectiveness and efficiency, and we discuss the results.

4 Choosing the Right Partners

First of all, when a mediator (say M) decides to cooperate with other agents, it must choose what are its partners. Indeed, it can choose to cooperate with other mediators as well as directly cooperate with information sources (see Fig. 3.b). Let us analyse such strategies focusing on efficiency and effectiveness. We evaluate the efficiency by means of the number of messages exchanged during the cooperation. We also evaluate the redundancy (of schemas) inside the information system that such strategies produce. Recall and precision ratios are used in order to evaluate the effectiveness.

Mediators. When M decides to cooperate with other mediators, it asks them to rewrite a query according to their own mediated schemas. If they succeed, M can store the result as new views in its mediated schema. This strategy can be convenient (*efficient*) when *the number of mediators is much less than the number of sources*, i.e. the number of messages to exchange is small. Nevertheless, we have to take into account the possibility that a sort of “*chain reaction*” occurs, i.e., the mediators involved by M decide to cooperate with other mediators, as well, and so on. On the other hand, having few mediators means that each mediator usually has information about a large number of sources and, therefore, can be helpful to cooperate with it. In order to evaluate the effectiveness, let us consider a mediator M that sends to other mediators the original query, and asks

[4] For example, $\forall pub.agents \sqcap \forall pub.db \sqcap \forall pub.\forall type.\{“Trans''\} \sqcap \forall pub.\forall publisher.\{“ACM''\}$ that involves sources w_2, w_3.

them for receiving a rewriting of the query. We have to evaluate the possibility that the involved mediator (say N) may send a wrong rewriting. This fact has harmful consequences on the precision, as stated by the following theorem.

Theorem 1. *Let $S_1, \ldots, S_m$ be m information sources. Let $\mathcal{V}$ be a view of $S_1, \ldots, S_m$. Let M, N be two mediators such that M interacts with N when M fails. Let $\mathcal{C}_n(M)$ be the mediated schema of M after n interactions with N. Then*

$$\begin{array}{ll} \textit{Recall:} & \lim_{n\to\infty} \frac{card(\mathcal{C}_n(M)\cap\mathcal{V})}{card(\mathcal{V})} \leq 1 \\ \textit{Precision:} & \lim_{n\to\infty} \frac{card(\mathcal{C}_n(M)\cap\mathcal{V})}{card(\mathcal{C}_n(M))} \quad \textit{is indeterminate} \end{array}$$

In the above theorem, $\mathcal{V}$ denotes the information need of a given consumer. Such a theorem states that when the mediator N sends a rewriting that is part of $\mathcal{V}$, the recall and the precision grow, otherwise the recall does not change and the precision decreases. This means that is crucial the choice of what and how many mediators to cooperate with. Concerning this fact, M can cooperate with the following mediators (see also Fig. 3.b):

– *All the mediators* of the system.
– *Failing mediators*, i.e., mediators responsible of the local failure in query evaluation. This strategy takes into account changes of schemas in mediators that act as sources for M.
– *Succeeding mediators*, i.e., mediators that successfully cooperated with M in the past. This strategy is based on the idea that the information needs of a given consumer usually involve the same sources and mediators. This is a sort of *locality principle* that can be applied when a failure in query rewriting occurred.
– *New mediators*, i.e., recently added mediators or just mediators that have never been in touch with M[5]. In contrast with the previous strategy, this strategy may be useful when the usual sources and mediators do not help.

The first strategy may be quite expensive, but it allows M to succeed when there exists at least a mediator that has the right solution. The other strategies are more efficient (they involve a small number of mediators), but they do not assure that M is able to find the mediator with the right answer. The cooperation with other mediators has another possible disadvantage: asymptotically all the mediators may have the same mediated schemas, i.e. this strategy produces some redundancy. This result is a consequence of the following theorem.

Theorem 2. *Let M, N be two mediators such that M interacts with N when M fails. Let $\mathcal{C}_n(M)$ be the mediated schema of M after n interactions with N. Let $\mathcal{C}(N)$ be the mediated schema of N such that it does not change while N interacts with M. Then*

$$\lim_{n\to\infty} \frac{card(\mathcal{C}_n(M)\cap\mathcal{C}(N))}{card(\mathcal{C}(N))} = 1$$

[5] This strategy is made possible by *facilitators* of multi-agent systems, i.e., agents that help on retrieving other agents

The above theorem states that the mediated schema of M converges to the mediated schema of N when M cooperates with N. This seems to suggest that when the mediator cooperates with other mediators it should directly ask data instead of the rewritten query. This would allow any mediator to maintain its expertise and avoid redundancy.

Example 5. In the situation of Example 4, let us suppose that the mediator M chooses to cooperate with a mediator N which has the following views in its mediated schema.

$$\begin{aligned} &\langle\ \forall pub.acm\_tods,\ \langle \forall pub.\forall booktitle.\{``TODS''\} \sqcap \forall pub.acm\rangle, && (1)\\ &\qquad \langle(\forall pub.\forall booktitle.\{``TODS''\}, w_4), (\forall pub.acm, w_4)\rangle\ \rangle \\ &\langle\ \forall pub.acm\_tocl,\ \langle \forall pub.\forall booktitle.\{``TOCL''\} \sqcap \forall pub.acm\_trans\rangle, && (2)\\ &\qquad \langle(\forall pub.\forall booktitle.\{``TOCL''\}, w_4), (\forall pub.acm\_trans, w_1)\rangle\ \rangle \end{aligned}$$

The mediator decomposes the query Q_1 (see Example 2) and sends each component to N. For example, when N receives $\forall pub.acm$, it looks for maximally contained concepts and sends to M both Views 1 and 2. w_1 is unavailable (see Example 4), therefore M retains as a new view only View 1. Notice that, View 2 does not increase the recall ratio and decreases the precision. Moreover, after the application of this strategy, M and N have one more view in common; therefore the redundancy is increased.

Information sources When M decides to directly cooperate with information sources, it asks them to rewrite a query according to their own local schemas, i.e., they classify the query and return the subsumed concepts. If they succeed, M can store the result as new views in its mediated schema. This strategy can be convenient (*efficient*) when *the number of mediators is comparable to the number of sources*, i.e. M has no advantages, in terms of messages to exchange, in cooperating with other mediators. Furthermore, M does not risk any "chain reaction" effect, since a source does not need any further cooperation in order to formulate its answer. In order to evaluate the effectiveness, let us consider a mediator M that sends to information sources the original query, and receives the rewriting of the query (and eventually the data). This strategy guarantees a given consumer that the mediator M converges to the consumer's information need. Indeed it is possible to prove the following theorem.

Theorem 3. *Let $S_1, \dots, S_m$ be m information sources. Let $\mathcal{V}$ be a view of $S_1, \dots, S_m$. Let M be a mediator such that M interacts with $S_1, \dots, S_m$ when it fails. Let $\mathcal{C}_n(M)$ be the mediated schema of M after n interactions with $S_1, \dots, S_m$. Then*

$$\begin{aligned} &\textit{Recall:} && \lim_{n\to\infty} \frac{card(\mathcal{C}_n(M) \cap \mathcal{V})}{card(\mathcal{V})} = 1 \\ &\textit{Precision:} && \lim_{n\to\infty} \frac{card(\mathcal{C}_n(M) \cap \mathcal{V})}{card(\mathcal{C}_n(M))} = 1 \end{aligned}$$

The above theorem states that the mediator will asymptotically satisfy the information need of a given consumer (denoted by $\mathcal{V}$). Indeed, under the hypothesis that no name heterogeneity occurs (see Section 2), it is impossible that a source has wrong schemas. An information source has either the right answer or no answer at all. This means that the choice of sources to cooperate with is crucial only for efficiency of the system, i.e. how fast the mediator's schema converges to user's needs. For what concerns this fact, M can cooperate with the following sources (see also Fig. 3.b):

– *All the sources* of the system.
– *Failing sources*, i.e., sources responsible of the local failure in query evaluation. This strategy takes into account changes of sources' schema that affect the mediated schema of M.
– *Succeeding sources*, i.e., sources that successfully cooperated with M in the past. This strategy is based on the idea that when a user has a new information need (i.e., a failure in query rewriting occurred), the new need usually involves the same sources of the past (*locality principle*).
– *New sources*, i.e., recently added sources or sources that have never been in touch with M. In contrast with the previous strategy, this strategy may be useful to take into account new sources and when the usual sources and mediators do not help.

The first strategy is expensive, but it allows M to find all the available information that satisfies the given consumer's query. The other strategies are more efficient, but they do not assure that M is able to find the source with the right answer.

Example 6. In the situation of Example 4, let us suppose that the mediator chooses to look for new sources and cooperate with them. Let us suppose that w_4 is a new information source which contains ACM publications and has the following views in its conceptual schema:

$$\begin{aligned} acm\_ccs &\doteq \forall booktitle.\{``CCS''\} \sqcap \forall type.\{``Proc''\} \sqcap acm \\ acm\_jacm &\doteq \forall booktitle.\{``JACM''\} \sqcap journal \sqcap acm \\ acm\_tods &\doteq \forall booktitle.\{``TODS''\} \sqcap acm\_trans \sqcap acm \end{aligned}$$

The mediator decomposes the query Q_1 and sends each component to w_4. For example, when w_4 receives $\forall pub.acm$, it looks for maximally contained concepts and obtains $\{\forall pub.acm\_ccs, \forall pub.acm\_jacm, \forall pub.acm\_tods\}$. When the mediator receives the answer, it retains the following new view:

$$\begin{aligned} &\langle \forall pub.acm, \langle \forall pub.acm\_ccs\ ,\ \forall pub.acm\_jacm\ ,\ \forall pub.acm\_tods \rangle, \\ &\qquad \langle (\forall pub.acm\_ccs, w_4), (\forall pub.acm\_jacm, w_4), (\forall pub.acm\_tods, w_4) \rangle \ \rangle \end{aligned}$$

5 Choosing the Right Queries

Cooperation strategies can also be classified according to queries (see Fig. 3.b). Indeed M can send the *original query* as received by the consumer. This is a strategy that takes into account new user's needs. M can also send the *retrieved views* (if any), i.e. the views subsumed by the consumer's query. This is a strategy that takes into account the need of maintaining consistent the mediator's schema, when a failure in query evaluation occurred. Finally, the mediator can send the *reformulated query* (if any). This is a strategy for maintaining consistent the mediated schema, as well. With the last two strategies the "approximation of the solution grows", as stated by the following theorem.

Theorem 4. *Let Q be the original query. Let Q' be one of the views retrieved by M. Let $Sol_M(Q')$ be one of the reformulations of Q' used by M as rewriting of Q. Let N be the mediator or source that cooperates with M. Let $Sol_N(x)$ be the query rewritten by N if N receives x as input query. Then*

$$Sol_N(Q) \sqsubseteq Q$$
$$Sol_N(Q') \sqsubseteq Q' \sqsubseteq Q$$
$$Sol_N(Sol_M(Q')) \sqsubseteq Sol_M(Q') \sqsubseteq Q' \sqsubseteq Q$$

As a consequence, the solution returned by N is more distant from Q than Q' when M does not send the original query.

Example 7. In the situation of Example 6, let us suppose that the mediator chooses to send one of the retrieved views (e.g., $I \doteq \forall pub.acm\_journal$) instead of the original query. In this situation, when w_4 receives I, it obtains $\{\forall pub.acm\_jacm, \forall pub.acm\_tods\}$, that is a subset of the result in Example 6.

If the mediator chooses to send one of the rewriting of I (e.g., $\forall pub.acm\_trans$), then w_4 returns $\{\forall pub.acm\_tods\}$, that is a subset of both the previous result and the result in Example 6.

Cooperation strategies can also be classified depending on the fact that the query can be sent as it is or be decomposed in basic components. For what concerns decomposition, when all the sources and mediators use C-Classic and decompose queries before their classification (see Section 2.1), we have that the following theorem holds:

Theorem 5. *Let $Q_1, \ldots, Q_n$ be queries. Let $Q = f(Q_1, \ldots, Q_n)$ be the user's query. Let N be the mediator or source that cooperates with M. Let $Sol_N(x)$ be the query rewritten by N if N receives x as input query. Then*

$$Sol_N(f(Q_1, \ldots, Q_n)) = f(Sol_N(Q_1), \ldots, Sol_N(Q_n))$$

The theorem above states that M does not need to decompose a query before its sending, since the query is directly decomposed by partners[6]. This suggests that decomposition is convenient (efficient) only if M failed to reformulate / evaluate just some components of the query.

6 Choosing the Right Answers

Finally, cooperation strategies can be classified according to answers (see Fig. 3.b). M can ask for rewriting the query. Its goal is to update its mediated schema with a rewritten query obtained from its collaborators. M can also

[6] Otherwise, $Sol_N(f(Q_1, \ldots, Q_n)) \sqsubseteq f(Sol_N(Q_1), \ldots, Sol_N(Q_n))$

Table 1. A comparison of different cooperation strategies

Partner	Type of Answer	New view in M's schema	Received Data	Efficiency			Effectiveness (Convergency)	
				No. of messages		Redundant		
				1st. time	Next time		Recall	Precision
Source (S_j)	Rewriting	$\langle Q, Sol_{S_j}(Q), S_j\rangle$	$I_{S_j}(Sol_{S_j}(Q))$	$4s$	$2s$	No	1	1
"	Data	$\langle Q, Q, S_j\rangle$	$I_{S_j}(Sol_{S_j}(Q))$	$2s$	$2s$	No	1	1
Mediator (N_i)	Rewriting	$\langle Q, Sol_{S_j}(Q), S_j\rangle$	$I_{S_j}(Sol_{S_j}(Q))$	$2m + 2s'$	$2s'$	Yes	≤ 1	No
"	Data	$\langle Q, Q, N_i\rangle$	$I_{S_j}(Sol_{S_j}(Q))$	$2m + 2ms'$	$2m + 2ms'$	No	≤ 1	No

ask for data that answer the query. In such a situation, it stores in the mediated schema the addresses of the mediators/sources that answered. The analysis of such strategies depends on partners; the results are depicted in Table 1.

We assume that collaborators (sources $S_1, \ldots, S_s$ and mediators $N_1, \ldots, N_m$) do not change while the strategy is in progress. We also suppose that N_i has $\langle Q, Sol_{S_j}(Q), S_j\rangle$ (with $j = 1, \ldots, s' \leq s$) in its mediated schema. For each strategy, Table 1 reports the new view that M stores in its mediated schema, the set of instances related to the reformulation, the upper bound of the number of messages for obtaining instances of Q (when M receives Q the first time and the next time), the results about redundancy (see Theorem 2), and the asymptotic behavior of recall and precision (see Theorems 1 and 3). For what concerns effectiveness, notice that different choices about answers do not affect the resulting data. For what concerns efficiency, when we consider cooperation with mediators, it seems more efficient to ask for a rewriting. Indeed, M interacts with N_i only the first time, the next time M directly interacts with S_j. On the other hand, the strategy of requiring data does not produce redundancy, but it is less efficient. When we consider the cooperation with sources, the most efficient strategy is that one requiring data.

Example 8. In the situation of Example 5, let us suppose that M asks for data. When it receives the answer from N, it retains the following view:

$$\langle \forall pub.acm, \langle \forall pub.acm\rangle, \langle(\forall pub.acm, N)\rangle\rangle$$

instead of View 1 of Example 5. Therefore, every time M selects this view, it has to request data to N that will request data to w_4. With View 1 retained in Example 5, M can directly request data to w_4.

7 Discussion

We considered the problem of rewriting queries by means of cooperation with other mediators and sources. This allows us to have dynamic mediated schemas instead of static ones. Thanks to this approach, the mediator is able to be updated when sources and consumers are added/removed or change their schemas/needs. This operation is performed only when a consumer sends a query that fails and not every time something changes. This allows us to avoid of overloading the distributed information system and the network with expensive consistency maintenance operations. In this approach, the choice of the right cooperation strategy is crucial. We analyse several possible strategies and discussed their advantages. From such an analysis arises two possible scenarios. *First*, when the number of sources is comparable to the number of mediators, the most efficient and effective strategy is to directly cooperate with sources. This seems to suggest that mediators can be fruitfully used to solve name heterogeneity and integrate data but not to solve schema heterogeneity. *Second*, when the number of sources is very large (e.g., when sources are web servers), the most efficient strategy is the cooperation with other mediators. Indeed, each mediator should be able to focus on an appropriate (and small) subset of available sources, reducing transmission volume and costs. In this situation, mediators act as sources for another mediator. Notice that the presence of different mediators is specifically useful when there are several consumers with different information needs.

References

1. V. Arens, C. Y. Chee, C.-N. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993. 123
2. C. Beeri, A. Y. Levy, and M.-C. Rousset. Rewriting Queries Using Views in Description Logics. In *Proc. of the 16th ACM Symposium on Principles of Database Systems (PODS)*, pages 99–108, New York, NY, May 12–14, Tucson, Arizona 1997. ACM Press. 123, 125, 126
3. A. Borgida and P. F. Patel-Schneider. A Semantics and Complete Algorithm for Subsumption in the CLASSIC Description Logic. *Journal of Artificial Intelligence Research*, 1:277–308, 1994. 123, 124
4. C. Diamantini and M. Panti. A Conceptual Indexing Method for Content-Based Retrieval. In A. M. Tjoa, A. Cammelli, and R. R. Wagner, editors, *Tenth International Workshop on Database and Expert Systems Applications (DEXA'99)*, Florence, Italy, 1–3 September 1999. IEEE Computer Society. 125
5. O. M. Duschka and M. R. Genesereth. Infomaster - An Information Integration Tool. In *Proc. of the International Workshop on Intelligent Information Integration*, Freiburg, Germany, September 1997. 123
6. T. A. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *AAAI Spring Symposium on Information Gathering*. AAAI, 1995. 123
7. M. Panti, L. Spalazzi, A. Giretti. A case-based approach to information integration. In *Proceedings of the 26th International Conference on Very Large Databases*, Cairo, Egypt, 10–14 September 2000. 124, 125, 126
8. A. Nica and E. A. Rundensteiner. DIIM: A Foundation for Translating Loosely-Specified Queries into Executable Plans in Large-Scale Information Systems. In *Proc. of the Second IFCIS International Conference on Cooperative Information Systems (CoopIS'97)*, pages 213–222, Kiawah Island, South Carolina, USA, June 24-27 1997. IEEE-CS Press. 124
9. J. D. Ullman. Information Integration Using Logical Views. In *Proceedings of the International Conference on Database Theory*, pages 19–40, 1997. 123, 125, 126
10. G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer Magazine*, 25:38–49, March 1992. 123

Planning and Optimizing Semantic Information Requests Using Domain Modeling and Resource Characteristics

Shuchi Patel and Amit Sheth

Large Scale Distributed Information Systems (LSDIS) Lab
Department of Computer Science, University of Georgia, Athens, GA 30602
{shuchi,amit}@cs.uga.edu
http://lsdis.cs.uga.edu

Abstract. The focus of information integration systems providing single access to data from distributed, diverse, and autonomous sources (including web-based sources) has changed from syntax and structure to semantics, allowing more meaningful integration of data. InfoQuilt[1] goes one step further to support knowledge discovery by providing users with tools to analyze the data, understand the domains and relationships between them, and explore new potential relationships. It provides a framework to model the semantics of domains, complex semantic relationships between them, characteristics of available sources, and provides an interface to specify information requests that the system can "understand". This thesis focuses on the use of knowledge about domains, their relationships, and sources to efficiently create practical execution plans for such semantic information requests.

1 Introduction

The amount of "data" available has exploded immensely in the last few years. The sources of this data, including the traditional file systems, databases, web-based sources, etc., are autonomous repositories that were developed independently. They therefore have different models for same domains. Most tools available to search information from web-based sources provide pure keyword based search that is not powerful enough to describe a user's information need. For instance, how would you describe the following request to a search engine such that it knows precisely what you are asking for? *"Find information about all earthquakes that occurred after 1990 in India resulting in death of more than 1000 people."* Also, the results generated have a low precision and recall. The condition of finding too much, and often irrelevant, data in response to any information need, whether using a search or browsing, is known as the problem of *information overload.*

[1] One of the incarnations of the InfoQuilt system, as applied to the geographic information as part of the NSF Digital Library II initiative is the ADEPT-UGA system [1]

© Springer-Verlag Berlin Heidelberg 2001

Providing access to diverse multiple sources via a common interface has been an interesting research problem called *information integration*. Sheth [19] describes different types of heterogeneities to be dealt with and related issues for achieving interoperability. Use of metadata, especially domain specific metadata, to address this issue is interesting and has gone a long way. [20] presents several examples. However, the need to semantically relate data to make sense out of it is what makes the problem more interesting and challenging. Consider the following information request. *"Find all nuclear tests conducted by USSR or US after 1970 and find all earthquakes that could have occurred as a result of the tests."* Among other things, the system needs to understand how a nuclear test can cause an earthquake. This requires understanding of semantics of the domains and the interaction between them. Structured databases that mainly focus on syntax and structure of data do not support this.

Very frequently, users use functions to post-process and analyze data (e.g. statistical analysis techniques). Of special interest are simulation programs that are used for forecasting, extrapolation, etc. Consider the following request. *"Forecast the urban growth pattern in the San Francisco bay area for the next 10 years in time steps of 1 year based on roads, slopes and vegetation patterns in the area."*

Some of the important issues in an information integration system are:

– Modeling the domains and relationships between them
– Resolving heterogeneities between the sources [19]
– A powerful interface to specify information requests (beyond traditional keyword queries and queries against structured database as in SQL) that can semantically describe the information need
– Scalability of the system
– Efficient planning and optimization

InfoQuilt addresses several of these issues. Several other research efforts such as [4], [13], [15], and [10] have developed approaches to represent the semantics of domains and the characteristics of sources. Additionally, InfoQuilt provides frameworks to model complex inter-domain relationships and information requests that capture the semantics of an information need. The vision of the InfoQuilt system is different from most other information integration systems. Our goal is to allow users to analyze the data available from a multitude of diverse autonomous sources, gain better understanding of the domains and their interactions and determine information leading to decision making via analysis of potential semantic relationships between the data provided by different sources. This support for knowledge discovery is a novel feature of InfoQuilt.

This paper focuses on planning and optimization of information requests, which are known as Information Scapes or IScapes in InfoQuilt. The algorithms described have been implemented and the examples presented have been tested. Given an IScape, the system creates an execution plan consisting of queries against relevant sources and steps to integrate the data. Creation of high-quality near-optimal plans is crucial due to the following main reason. Web sources constitute a large portion of available sources. Retrieving data from them over the network using wrappers is slow relative to a source that is a database. The query execution is therefore relatively slow. However, certain semantic optimizations allow us to choose one execution plan over another so that the execution is faster. The following are key features of our algorithm:

– Efficient source selection - excluding sources that will not provide useful results or will provide redundant data
– Generation of executable plans which respect the limitations of resources
– Use of sources in conjunction to be able to use resources with query capability limitations and missing data
– Integration of data retrieved from the sources selected, including relationship evaluation and post-processing (e.g. evaluating functions and running simulations)

Section 2 briefly describes how the knowledge base of the system is represented, how IScapes are represented, and how InfoQuilt supports an environment for knowledge discovery. Section 3 describes planning of IScapes and optimization of the execution plans. Section 4 discusses the runtime architecture of the system and IScape execution. We compare InfoQuilt with other systems in sect 5. Section 6 presents our conclusions and directions for future work.

2 Background

In a typical scenario in using the InfoQuilt system, an administrator first identifies the domains of interest and models them. Section 2.1 describes domain modeling. Next, he models complex relationships between the domains as described in Sect. 2.2. He then identifies data sources for those domains, creates their wrappers, and models them as described in Sect. 2.3. A distinguishing feature of InfoQuilt is its ability to use functions. We describe them in sect 2.4. The knowledge about ontologies, relationships, resources and functions forms the *knowledge base* of the system. The administrator uses a graphical tool called the *Knowledge Builder* (KB) to easily create and maintain the knowledge base. Users can then create IScapes as discussed in Sect. 2.5 using a visual tool called the *IScape Builder* (IB). Section 2.6 shows how InfoQuilt can help users discover knowledge. Details left out in this paper due to space restriction can be found in [17]. Additional detailed discussion on the knowledge base, KB, IScapes, IB, and the support for knowledge discovery can be found in [21].

2.1 Domain Modeling

One form of heterogeneity between different sources that needs to be resolved is the difference in their views of the domain. InfoQuilt uses ontologies for this. An *ontology* captures useful semantics of the domain such as the terms and concepts of interest, their meanings, relationships between them, and characteristics of the domain. The domain characteristics are described as a set of domain rules and functional dependencies. We will use the terms ”ontology” and ”domain” interchangeably in the rest of the paper unless explicitly specified.

Example 1. Consider the domain of nuclear tests. Some related terms are test site, explosive yield, seismic body wave magnitude, type (underground, atmospheric, etc.), latitude, longitude, etc. The magnitude of a test is in the range 0 to 10.[2] We represent this as domain rules. Also, given a test site, the latitude and longitude are the same. We represent this as a functional dependency (FD).

```
NuclearTest( testSite, explosiveYield, waveMagnitude, testType,
             eventDate, conductedBy, latitude, longitude,
             waveMagnitude > 0, waveMagnitude < 10,
             testSite -> latitude longitude );
```

We will use the above notation to represent ontologies in the paper. □

2.2 Inter-ontological Relationships

Real world entities are related to each other in various ways. These relationships can be simple , for e.g., a car "*is a*" vehicle, or complex, for e.g., an earthquake "*causes*" a tsunami. InfoQuilt is novel in its ability to model such relationships.

Example 2. Consider the relationship between a nuclear test and an earthquake. We use the `NuclearTest` ontology from e.g. 1 and model earthquake as follows:

```
Earthquake( eventDate, description, region, magnitude, latitude,
            longitude, numberOfDeaths, damagePhoto,
            magnitude > 0 );
```

We can say that a nuclear test could have "*caused*" an earthquake if the earthquake occurred "*some time after*" the test was conducted and "*in a nearby region*". Here, "*some time after*" and "*in nearby region*" are user-defined functions used as specialized operators as described in Sect. 2.4. For now, assume that there are two functions called `dateDifference` and `distance` that compute the difference between two dates and the distance between two locations (specified by their latitudes and longitudes) in miles respectively. We can now represent the relationship as follows:

```
NuclearTest Causes Earthquake <=
dateDifference(NuclearTest.eventDate, Earthquake.eventDate) < 30
AND distance(NuclearTest.latitude, NuclearTest.longitude,
               Earthquake,latitude, Earthquake.longitude) < 10000
```

The values 30 and 10000 are arbitrary. □

[2] The body wave magnitude does not have upper or lower bounds. However, all nuclear tests as well as earthquakes measured to date had a magnitude less than 10 and those with magnitudes less than 0 are very small. We use these bounds solely to demonstrate how the domain rules can be modeled and used.

2.3 Source Modeling

The system needs to resolve heterogeneities between the sources. This is done in part by creating an ontology for each domain and describing the sources in terms of them. This approach is known as source-centric approach. Since the sources are modeled completely independent of each other, they can be easily added, removed and re-modeled without changing the ontologies. The source models describe their characteristics and query limitations in terms of the following:

Resource Rules: Resource or Data Characteristic (DC) rules describe conditions that always hold true for data retrieved from resource.

Example 3. Consider a resource that lists earthquakes in California after January 1, 1980. We can specify this by the rules:

```
region = "California"
eventDate >= "January 1, 1980"
```

□

Local Completeness: Local Completeness (LC) rules describe a subset of the domain for which the resource is known to have complete information.

Example 4. Ontology `DirectFlight` models direct flights from one US city to another. The database resource at Atlanta Airport provides data on *every* flight departing from and arriving at Atlanta. We specify this using the LC rules:

```
toCity = "Atlanta"
fromCity = "Atlanta"
```

□

Binding Patterns: Web sites that allow users to search their database(s) via HTML forms using CGI scripts, servlets, etc. often require the user to provide values for certain parameters. The query answering system needs to provide values for these parameters to use it. It is crucial for the system to be aware of these limitations. These are represented as binding patterns (BP).

Example 5. The AirTran Airways (*http://www.airtran.com*) web site requires the user to specify source, destination, dates of travel, etc. We represent this as the following BP:

```
[toCity, toState, fromCity, fromState,
departureMonth, departureDay]
```

□

2.4 Functions and Simulations

Example 2 demonstrates use of two functions to create special operators used to define a relationship between `NuclearTest` and `Earthquake`, which cannot be expressed by using only relational and logical operators. Another use of functions is to post-process data. For e.g., use of various statistical analysis techniques to analyze data is very frequent. Of particular interest, are simulations. For e.g., researchers in the field of Geographical Information Systems (GIS) use them to extrapolate patterns of urban growth, deforestation, etc. based on models. These are available as independent programs. They can be used over the data retrieved from available sources (assuming that the system has access to appropriate sources) and the result can be presented to the user.

InfoQuilt views such functions and simulations as important sources of related information. It maintains a declarative description of the functions in a *Function Store*, a component of its knowledge base. The administrator is responsible for providing an implementation for them.

2.5 Information Scapes (IScapes)

InfoQuilt is unique in its ability to answer complex information requests known as *Information Scapes* or *IScapes*. Since they are specified in terms of the components of the knowledge base of the system, they are better able to model the semantics of a user request and the system is able to "understand" them.

Example 6. Consider the following IScape (described here as text).

"Find all nuclear tests conducted by USSR or US after 1970 and find information about earthquakes that could have occurred due to them."

Of specific interest is that the system needs to understand what the user means by "*earthquakes that could have occurred due to these tests*". This is where knowledge about the inter-ontological relationship "nuclear test causes earthquakes" available in the knowledge base of the system is useful. □

IScape also abstracts the user from the characteristics, structure, access mechanisms, etc. of the actual data sources available to the system (eventually used to answer the IScape), and their heterogeneities.

We provide a graphical toolkit, known as *IScape Builder*, that allows users to easily create IScapes step-by-step, execute them and further analyze the data. See [21] for details.

2.6 Support for Knowledge Discovery

InfoQuilt helps users discover knowledge by providing tools to analyze the data available from diverse distributed sources, gain understanding of the domains and their relationships, explore possibilities of new relationships, explore trends to support such potential relationships, and provide further insight into additional aspects about existing relationships. A detailed discussion with examples appears in [21].

3 Planning and Optimization

We now describe the creation of execution plans for IScapes and optimizations possible due to the knowledge base that InfoQuilt maintains. The sources are described in terms of the ontologies. Hence, source descriptions are similar to views defined on them. The problem of creating an execution plan is thus closely related to the problem of answering queries using views [23], [12], [7]. It was shown to be NP-complete in [12]. The algorithm we describe here uses domain and source characteristics and is therefore tractable and efficient.

A more detailed explanation of plan generation can be found in [17]. We use the following example IScape and show how its execution plan is generated.

”*Find nuclear tests conducted by USSR after January 1, 1980 with magnitude greater than 5 and earthquakes that could have occurred due to these tests.*”

We use the ontologies `NuclearTest` and `Earthquake` and the relationship between them `NuclearTest Causes Earthquake`. We use a similar notation for sources as the one we use for ontologies. We prepend `[dc]` or `[lc]` to distinguish between DC and LC rules. Following sources are available to the system.

```
NuclearTestsDB(testSite, explosiveYield, waveMagnitude,
               testType, eventDate, conductedBy,
               [dc] waveMagnitude > 3,
               [dc] eventDate > "January 1, 1985");
NuclearTestSites(testSite, latitude, longitude);
SignificantEarthquakesDB(eventDate, description, region,
                         magnitude, latitude, longitude,
                         numberOfDeaths, damagePhoto,
                         [dc] eventDate > "January 1, 1970");
```

`NuclearTestsDB` is a database of nuclear tests of magnitude greater than 3 conducted after January 1, 1985. `NuclearTestSites` provides exact locations of nuclear test sites in terms of latitudes and longitudes. `SignificantEarthquakesDB` is a database of significant earthquakes occurring after January 1, 1970.

Step 1 (Check Semantic Correctness). The first step is to check if the IScape is semantically valid by comparing the IScape constraint with the domain rules defined on the ontologies involved. The constraint in our IScape is:

```
NuclearTest.waveMagnitude > 5 AND NuclearTest.conductedBy = "USSR"
AND NuclearTest.eventDate > "January 1, 1980"
AND dateDifference(NuclearTest.eventDate,
                   Earthquake.eventDate) < 30
AND distance(NuclearTest.latitude, NuclearTest.longitude,
             Earthquake,latitude, Earthquake.longitude) < 10000
```

The system maps the relationships used in the IScape into sub-constraints. Comparing the constraint with the domain rules defined on the ontologies, we see that the IScape is semantically valid.

If the IScape enquired about nuclear tests with magnitude greater than 12 instead of 5, the domain rule "`waveMagnitude < 10`" would imply that the IScape is semantically invalid. This aspect of our algorithm is supported by various systems that support semantic integrity constraints.

Step 2 (Source Selection). Next, the planner uses domain rules and FDs of ontologies and DC rules, LC rules, and BPs of the sources to select relevant sources by applying the following rules for each ontology:

Locally Complete Sources : If there exists a source that is locally complete for some subset A of the domain of the ontology such that the part of the IScape's result that comes from that domain is a subset of A, other sources need not be used (unless the selected source has some attributes missing or a BP) since the LC rule on the source implies that using any additional sources will not provide any extra information. No source in our example are locally complete.

Non Locally Complete Sources : If a locally complete source could not be found, then the planner considers all the sources. The sources whose DC rules falsify IScape's constraint are then eliminated since they will not return any useful results for the IScape (they will get filtered by the IScape constraint).

For our example, the planner selects all sources for both ontologies and none of them are eliminated. If `NuclearTestsDB` provided data for tests conducted *before* January 1, 1980, then the rule "`eventDate < January 1, 1980`" would falsify IScape's constraint and the source would be eliminated.

Binding Patterns (BP) : Planner then decides how the values for the BPs on resources, if any, will be supplied. It can be (a) from the IScape constraint, (b) from attributes in other ontologies or (c) by using another source (associate) of the ontology in conjunction to retrieve some arbitrary values for the attribute. None of the resources selected for our example have BPs. Refer to [17] for an example that uses resources with BPs.

Associate Resource to Supply Values for Missing Attributes : It is common to come across sources that do not provide all the attributes needed. If a source has one or more attributes missing, the planner can use the FDs on the ontology to couple it with an associate resource to retrieve values of those attributes for as many entities as possible. This is done by equating the values of attributes in the LHS of the FD to join the data retrieved from the main and associate resources. System can thus deduce more information by using sources in conjunction. The attributes are equated using a default equality computing function defined for it in the function store or an exact match if no function was defined. Use of functions is necessary as it is highly unlikely that all sources will have exact same values for the attributes for a given entitiy (syntactic heterogeneity) [11]. For e.g., a nuclear test site available from one source could be "Nevada Test Site, Nevada, USA" and that from another source could be "Nevada Test Site, NV, USA". The two are semantically equal but syntactically unequal. Functions can be used to compute a context-sensitive fuzzy equality.

`NuclearTestsDB` has two missing attributes - `latitude` and `longitude`. The planner uses the FD "`testSite -> latitude longitude`" and `NuclearTestSites` as an associate resource to retrieve their values. A function `testSiteEquals` is used to equate the values of `testSite` from the two resources. Most attributes are missing from `NuclearTestSites` and there is no FD that can be used to retrieve thier values. Hence, it is eliminated it as a primary source.

The plan may use a source more than once. For e.g., a source may be used as a primary source and an associate source. The planner identifies such sources and optimizes the plan by creating a single Resource Access node.

Figure 1 shows the plan for our IScape. It shows how some sub-conditions can be pushed to sources and how some sub-conditions may be eliminated using DC rules.

`NuclearTestsDB` is joined with `NuclearTestSites` to provide values for the missing attributes - `latitude` and `longitude`. There is no check for the sub-constraint `NuclearTest.eventDate > "January 1, 1980"` because only data on tests conducted after January 1, 1985 is available from `NuclearTestsDB`.

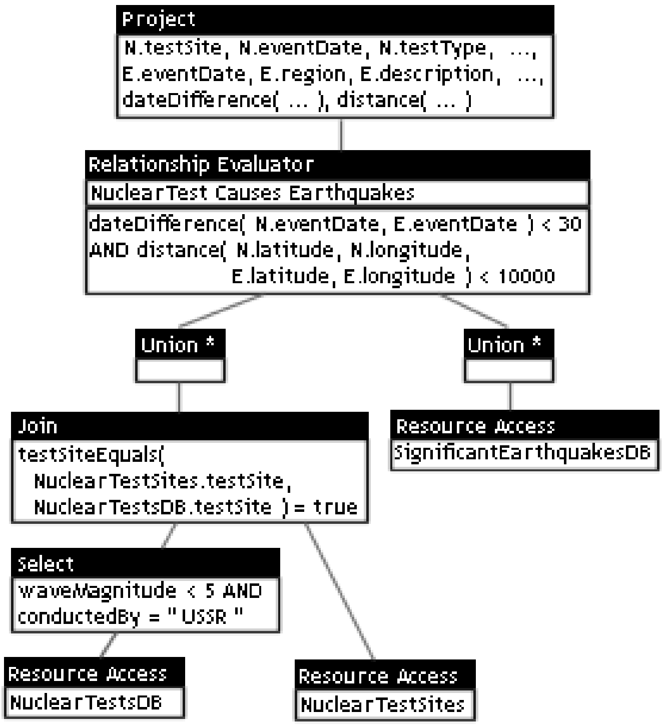

Project
N.testSite, N.eventDate, N.testType, ...,
E.eventDate, E.region, E.description, ...,
dateDifference(...), distance(...)
Relationship Evaluator
NuclearTest Causes Earthquakes
dateDifference(N.eventDate, E.eventDate) < 30
AND distance(N.latitude, N.longitude,
E.latitude, E.longitude) < 10000
Union *
Union *
Join
testSiteEquals(
NuclearTestSites.testSite,
NuclearTestsDB.testSite) = true
Resource Access
SignificantEarthquakesDB
Select
waveMagnitude < 5 AND
conductedBy = "USSR "
Resource Access
NuclearTestsDB
Resource Access
NuclearTestSites

Fig. 1. IScape Execution Plan

Step 3 (Integrate Data For Each Ontology). Next, the planner creates the nodes that describe how the data retrieved from the sources is to be integrated. The first step in doing this is to create unions of results retrieved from sources selected for each ontology in the IScape.

For this, it first adds a node to compute an intermediate union of data from those sources that do not retrieve values for their BP from attributes of other ontologies. A source (of another ontology) that needs values of some attribute(s) of this ontology for its BP can then retrieve them from the intermediate union. This avoids deadlocks. We use a * in Fig. 1 to denote a union node of this type. The planner then adds another union node that computes the final union of data from all sources for the ontology. At all stages of plan creation, if a node is found to be trivial, it is eliminated.

Step 4 (Supplying BP Values from Other Ontologies). After creating the unionsNext, the planner goes through updates all the BPSupplier nodes (used with for sources that retrieve BP values from other ontologies) in the plan by creating links from it to the check that there exists an intermediate union nodes for of the ontologies from which it retrieves BP values are to be retrieved. If it does not, the source that the BPSupplier node supplies BP values to cannot be used. If the intermediate unions do exist, the planner updates the plan to point to them. See [17] for details and example.

Step 5 (Relationship and Constraint Evaluations). Next, for each ontology, all the functions and sub-constraints that involve attributes of only that ontology are evaluated. The data from different ontologies should then be integrated by evaluating the relationships used in the IScape or by creating joins if some ontology is not a part of any relationship. All remaining functions and constraints (that involve attributes of multiple ontologies) can be evaluated next.

Step 6 (Aggregation and Final Projection). The results are then grouped as specified, aggregations are computed and group constraints are evaluated. Our IScape does not require grouping of results. Finally, the result can be projected on the operands (attributes, functions and aggregates) in the projection list of the IScape. This completes the generation of the execution plan for the IScape.

4 IScape Execution

This section discusses how an IScape submitted to the InfoQuilt system is processed. We start by describing the runtime architecture.

4.1 Runtime Architecture

InfoQuilt uses an agent-based brokering architecture shown in Fig. 2. Although the agent-based architecture itself is not unique (e.g., see [5]), the capabilities of the agents to support complex relationships, multiple ontologies and IScapes differentiate the system from previous agent-based systems. This work is built upon the work done in [18], [6], and [16]. The system provides a visual tool to monitor the execution. See [17] for details.

The Knowledge Agent acts as an interface to the knowledge base of the system represented by ”Knowledge” in Fig. 2. The Broker Agent is responsible for brokering requests from other agents and co-ordination of IScape processing. All agents interact through the Broker Agent. The Resource Agents provide a uniform interface to access all resources. There is one Resource Agent per resource in the system. They address issues related to access mechanisms, mapping data from the source views to global views of the system, etc. The processing of an IScape proceeds as follows:

1. The User Agent sends the Iscape to the Broker Agent for processing.
2. The Broker Agent sends the Iscape to the Planning Agent.
3. The Planning Agent creates a plan for the Iscape as described in Sect. 3, enquiring about the ontologies, relationships, etc. from the Knowledge Agent. If the IScape is semantically invalid, it informs the Broker Agent and aborts.
4. The Planning Agent returns the plan to the Broker Agent, which responds back to the User Agent skipping steps 5-7 if plan generation was aborted.
5. The Broker Agent then sends the plan to the Correlation Agent.
6. The Correlation Agent executes the plan. It accesses the necessary resources (through Resource Agents) through the Broker Agent. We describe it in detail in sect 4.2.

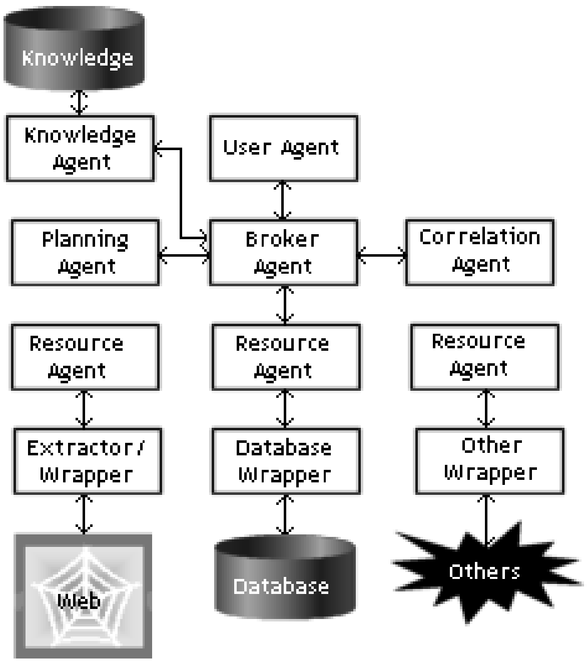


Fig. 2. InfoQuilt Runtime Architecture

7. The Correlation Agent returns the final result to the Broker Agent.
8. Finally, the Broker Agent forwards the result to the User Agent.

4.2 Multi-threaded Execution of IScape by Correlation Agent

IScape execution exploits the parallelism in the plan and is multi-threaded. The Correlation Agent starts by creating a *node processor* for each node in the plan. Each node processor is a separate thread. Every node is responsible for starting the node processors from which it expects any inputs (result sets). It then waits for all its inputs and starts processing as soon as they are available.

4.3 Execution of Functions and Simulations

Evaluation of functions (including execution of simulations) is done using Java's reflection package to pass arguments, invoke the function and retrieve the results back. The administrator is responsible for supplying the methods as static functions in Java classes.

However, simulations can be tricky. They could be executable files, scripts that need special interpreters, etc. Also, they may expect their inputs in different ways. Considering the diversity that can be encountered, the framework provided may not be sufficient for all kinds of simulation programs available. This needs further investigation. However, a large part of these can be supported using the current framework. For e.g., we use Clarke's Urban Growth Model [8] to predict urban growth in an area. The program is available as an executable and needs an input configuration file from which it reads in its parameters.

5 Related Work

SIMS [3] [4], TSIMMIS [10], Information Manifold [13], and OBSERVER [15] are some of the other related efforts. The goal of InfoQuilt is to allow users to query, analyze, reason about inter-domain relationships and analyze data available from multiple sources. However, most other systems focus only on retrieving and integrating "data" and not on the "exploring, understanding and knowledge discovery" aspects. The distinguishing features of InfoQuilt are:

– Ability to help in understanding domains and their complex relationships
– Support for functions and simulations
– Ability to model complex relationships
– Powerful semantic query interface (IScapes)

We briefly compare our vision and approach with other systems in this section. A detailed discussion appears in [17].

SIMS [3] [4] creates a model of the domain using a knowledge representation system and accepts queries in the same language. A major limitation of the system is that its mediator is specialized to a single application domain [2]. It also does not support inter-ontological relationships and functions.

OBSERVER [15] uses ontologies to describe information sources and inter-ontology relationships like synonyms, hyponyms and hypernyms across terms in different ontologies to translate a query specified using one ontology into another query that uses related ontologies. This approach of using relationships to achieve interoperability between the sources is interesting. However, it is limited to basic relationships.

TSIMMIS [10] uses a mediator-based architecture [22]. It uses Mediator Specification Language (MSL) to define mediators. The mediators are then generated automatically from these specifications. Since the MSL definitions need to be created manually, adding or removing information sources requires updating and recompiling the definitions. The mediator answers queries by relating them to pre-defined query templates. Hence, it can answer only a restricted set of queries. InfoQuilt has a dynamic planner that automatically considers newly added sources while planning IScapes.

Information Manifold [13] uses an approach similar to ours in that the user creates a world view, a collection of virtual relations and classes. The world view however does not capture semantics of the domains as InfoQuilt can using domain rules and FDs. Information sources are described to the system as a query over the relations in the world view. The user queries are also specified over relations in this world view. The sources can be specified to be either a subset of the domain or equal to a subset of the domain [14]. Hence, local completeness cannot be modeled precisely. IM uses capability records to capture query capability limitations of sources. The system arbitrarily selects a set of input parameters. This approach would not work if the source needs very specific combinations of attributes as input.

Duschka [9] present a comprehensive theory on planning queries in information integration systems. They focus on creation of maximally-contained query plans, defined as a plan that provides *all* the answers that are possible to obtain from the available sources. They show that it is necessary to consider plans with recursion in order to generate maximally-contained plans. Since a large number of information sources would be web sources that are slow relative to a source that is a database, execution of plans with recursion could be very slow. A query plan that does not necessarily return all the possible results may still be useful. We adopt this approach.

6 Conclusion

We discussed how InfoQuilt goes beyond the traditional querying techniques to provide access to and integrate data in a more semantic manner with the goal of providing a querying, analyzing and knowledge discovery environment. The key features of the system are:

– ability to model inter-domain relationships
– ability to use value-adding functions and simulations
– powerful query interface to describe complex information needs (IScapes)
– environment for knowledge discovery

We described the planning and optimization algorithms used in InfoQuilt. The planner is dynamic and flexible. It allows dynamic addition, deletion, and modification of ontologies, relationships, resources, and functions. The contributions of this paper are practical algorithms to efficiently select sources most appropriate to answer an IScape by excluding sources with redundant or no useful information in the context of the IScape, using sources in conjunction to retrieve more comprehensive information using the same set of available resources, generating executable plans that respect query capability limitations of the resources, integrating retrieved information by evaluating relationships and post-processing (evaluating functions and running simulations), and multi-threaded processing of the IScapes to exploit the parallelism in the plans.

Following are some of the planned future improvements. The Planning Agent could create backup plans that the Correlation Agent can switch to on failure. Simulations are currently supported using the framework used for functions. Simulation programs however are more complex and diverse in the kind of application (it could be an executable, a script, etc.), the method of accepting inputs, for e.g. as files from local file systems, etc. The framework currently used may not suffice as it assumes that they are available as separate functions (through wrappers if needed). Several simulations however exist as programs written using languages supported by special software, for e.g., ArcInfo, and hence, may be difficult to add to the Function Store. Different types of simulation programs therefore need to be explored more thoroughly to provide a framework better suited to support a large class of them, if not all.

References

1. Alexandria Digital Earth Prototype *http://alexandria.ucsb.edu/* 135
2. Y. Arens, C. Hsu and C. A. Knoblock. Query processing in the SIMS information mediator. In Austin Tate, editor, Advanced Planning Technology. The AAAI Press, Menlo Park, CA, 1996 146
3. J. Ambite, and C. Knoblock. Planning by rewriting: Efficiently generating high-quality plans. Proceedings of the 14th National Conference on Artificial Intelligence, Providence, RI, 1997 146
4. Y. Arens, C. A. Knoblock, and W. Shen. Query reformulation for dynamic information integration. Journal of Intelligent Information Systems, Vol. 6, pp. 99-130, 1996 136, 146
5. R. J. Bayardo Jr., W. Bohrer, R. Brice, et al. InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. In SIGMOD-97, pp. 195-206, Tucson, AZ, USA, May 1997 144
6. C. Bertram. InfoQuilt: Semantic correlation of heterogeneous distributed assets. Masters Thesis, Computer Science Dept, University of Georgia, 1998 145
7. S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim. Optimizing queries with materialized views. Proceedings of international conference on Data Engineering, 1995 141
8. Clarke's Urban Growth Model, Project Gigalopolos, Department of Geography, University of California, Santa Barbara *http://www.ncgia.ucsb.edu/projects/gig/ncgia.html* 146

9. O. M. Duschka. Query planning and optimization in information integration. Ph. D. Thesis, Computer Science Department, Stanford University, 1997 147
10. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, et al. The TSIMMIS approach to mediation: Data models and languages. In Proceedings of NGITS, 1995 136, 146, 147
11. M. Guntamadugu. MÉTIS: Automating metabase creation from multiple heterogeneous sources. Masters Thesis, Computer Science Department, University of Georgia, 2000 142
12. A. Y. Levy, A. O. Mendelzon, Y. Sagiv and D. Srivastava. Answering queries using views. Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of database systems, PODS-95, San Jose, CA, May 1995 141
13. A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. Proceedings of the 22nd International Conference on Very Large Databases, Bombay, India, September 1996 136, 146, 147
14. A. Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. International Journal on Intelligent Information Systems, pp. 121-143, 1995 147
15. E. Mena, A. Illarramendi, V. Kashyap, and A. P. Sheth. OBSERVER: An approach for query processing in global information systems based on interoperation across pre-existing ontologies. International Journal on Distributed and Parallel Databases, Vol. 8, No. 2, pp. 223-271, April 2000 136, 146, 147
16. K. Parasuraman. A multi-agent system for information brokering in infoQuilt. Masters Thesis, Computer Science Department, University of Georgia, 1998 145
17. S. Patel and A. Sheth. Planning and optimizing semantic information requests using domain modeling and resource characteristics. Technical Report, LSDIS Laboratory, University of Georgia, 2001. *http://lsdis.cs.uga.edu/proj/iq/iq_pub.html* 137, 141, 142, 144, 145, 146
18. D. Singh. An agents based architecture for query planning and cost modeling of web sources. Masters Thesis. Computer Science Department, University of Georgia, 2000 145
19. A. Sheth, Changing focus on interoperability: From system, syntax, structure to semantics. In M. Goodchild, M. Egenhofer, R. Fegeas, and C. Kottman, editors, Interoperating Geographic Information Systems, Kluwer Academic Publishers, 1999 136
20. A. Sheth and W. Klas, Editors. multimedia data management - Using metadata to integrate and apply digital media. Mc Graw-Hill, 1998 136
21. S. Thacker, S. Patel, and A. Sheth. Knowledge modeling for study of domains and inter-domain relationships - A knowledge discovery paradigm. Technical Report, LSDIS Laboratory, University of Georgia, 2001. *http://lsdis.cs.uga.edu/proj/iq/iq_pub.html* 137, 140
22. G. Wiederhold. Mediators in the architecture of future information systems. IEEE Computer, 25(3), pp. 38-49 147
23. H. A. Yang, and P. A. Larson. Query transformation for PSJ queries. Proceedings of 13th International Conference on Very Large Databases VLDB-87, Brighton, England, 1987, pp. 245-254 141

Integrating View Schemata Using an Extended Object Definition Language[⋆]

Mark Roantree[1], Jessie B. Kennedy[2], and Peter J. Barclay[2]

[1] School of Computer Applications, Dublin City University, Dublin, Ireland.
mark.roantree@compapp.dcu.ie
[2] School of Computing, Napier University, Edinburgh, Scotland.

Abstract. View mechanisms play an important role in restructuring data for users, while maintaining the integrity and autonomy for the underlying database schema. Although far more complex than their relational counterparts, numerous object-oriented view mechanisms have been specified and implemented over the last decade. These view mechanisms have served different functions: view schemata for object-oriented databases; object views of relational (and other) database systems, and the formation of federated schemata for distributed information systems. In the latter category there is still a significant amount of research required to construct a view language powerful enough to support federated views. Such a language (or set of languages) should support not only object views, but also a wrapper specification language for external information sources, and a set of restructuring and integration operators. Furthermore, with the advent of standard models and technologies such as CORBA for distribution, ODMG for storage, and XML for web publishing, these languages should be based upon, or cooperate with, these standards. In this research, we present a view mechanism which retains the semantic information incorporated in ODMG schemata, provide a set of operators which facilitate the restructuring and integration necessary to merge schemata, and provide wrappers to heterogenous systems such as legacy systems, ODBC databases, and XML data sources.

1 Introduction

The concept of a federation of databases [17] is one where heterogeneous databases (or information systems) can communicate with each other through an interface provided by a common data model. In our case, the common data model is the ODMG model, the standard model for object-oriented databases since 1993 [6]. The most common architecture for these systems is as follows: data resides in many (generally heterogeneous) information systems or databases; the schema of each Information System (IS) is generally translated to an O-O format, and this new schema is called the *component* schema; view schemata are defined (subsets of the component schema) that are shared with other systems; these view schemata are exported to a global or federated server where they are

⋆ Supported by Forbairt Strategic Research Programme ST/98/014

© Springer-Verlag Berlin Heidelberg 2001

integrated to form many global or federated schemata. Please refer to [4] for a fuller description of object-based federated database systems.

This paper is structured as follows: in the remainder of this section, we provide a motivation and discuss related work; in §2, ODMG view and wrapper languages are presented; in §3, two schemata are integrated using the view language; in §4, a description of the implementation takes place; and finally in §5, some conclusions are offered.

1.1 Background and Motivation

The OASIS project [11] dealt with the federation of healthcare systems using an ODMG model as canonical model. While the ODMG model provides the semantic power suggested in [16] for federated data models, and offered a standard which suited forward interoperability, it lacked fundamental features required for integrating information systems. The model must (at least) provide a view system, a facility for wrappers to external information systesms, and integration operators which facilitate the construction of federated schemata in (structurally) different ways. Furthermore, all of these features should be provided in the data definition language of the database: in this case ODMG's Object Definition Language. In the case of the view specifications, the view language must be capable of retaining as much semantic information as possible, and thus, a view which results in a single virtual class is not sufficient. Instead, the view should resemble a subschema that retains information regarding inheritance and relationships between classes. Furthermore, this view mechanism must be powerful enough to support the restructuring of the subschema specification for any schema integration step. The contribution of this research is to provide extensions to the ODMG metamodel to support virtual subschemata and their components; extend the Object Definition Language (ODL) to support wrapper and *semantically-rich* view specifications; and provide an architecture and services to support the transfer, integration and display of view schemata.

1.2 Related Work

Our research requires the development of an object-oriented view mechanism which provides operators capable of restructuring the object-oriented schema hierarchy. A number of view mechanisms have emerged for object-oriented databases in the past. Older views mechanisms concentrated on the construction of a single virtual class, a representation which loses vital (for the schema integration process) semantic information. Although later view mechanisms such as [15][7][18] provide a means of defining virtual schemata, they use proprietary object models (which hinder interoperability) and do not include valuable restructuring operators such as those described in [10]. Recent work which offers a view mechanism can be found in [1] where they also adopt the usage of modern standards (XML) to offer an object-based view mechanism.

2 View Language Syntax

In this section, the syntax of the ODLv language is presented by illustrating the important production rules and providing a brief overview of operations. Space restrictions prevent a detailed discussion of the language, although this can be found in [14]. The ODMG provides the Object Definition Language (ODL) for defining base schema: we have defined the ODLv specification language for views, and the ODLw specification language for wrappers.

The main subschema production (not shown here) contains a number of `SchemaSegment` productions which in turn contains productions to import base and virtual classes, restructure imported classes, and export those classes which comprise the view schema. In the following BNF expressions, assume `identifier` to be a single identifier; `qualifier_dcl` to be an identifier with a qualifier (eg. class.property); and `double_qualifier_dcl` to be an identifier with two qualifiers (eg. viewname.class.property).

In *definition 1*, some idea is provided of the types of operations permitted at the property level: renaming of properties; hiding of properties, and the derivation of new attributes. Note that the first two operations are for attributes *and* relationships, and the third property is for attributes only.

Definition 1: property_dcl:

```
"property"
    ( property_rename_dcl)?
    ( property_hide_dcl)?
    ( property_derive_dcl)?
```

Each class declarator contains three productions: the `class_operation` production, the `class_rename` production, and the `class_filter` production.

Definition 2: class_dcl:

```
"class"
( class_operation_dcl)?
( class_rename_dcl)?
( class_filter_dcl)?
```

The `rename` expression can be used to rename classes. The `class_filter_dcl` is used to include an OQL query for the chosen class, and this topic of virtual class extents is discussed fully in §3.2. Each operator inside the `operation` expression falls into one of two categories: restructuring and integration operators.

2.1 Restructuring Operators

The view mechanism contains five restructuring operators: `aggregate`, `expand`, `subclass`, `superclass`, and `flatten`. Of the five operators, the first two operators, `aggregate` and `expand`, deal with association relationships, and the remaining three operators, `subclass`, `superclass` and `flatten`, deal with hierarchical relationships. For space reasons we will discuss three operators: `aggregate`, expand and subclass. Of the remaining two, the superclass operator is used to create a new superclass from an existing class, and the flatten operator is used to *collapse* or *fold* a superclass into one of its subclasses.

Definition 3: class_aggregate_dcl:
identifier := "**aggregate**"
(identifier (,identifier)*)
"**from**" qualifier_dcl
("**as**" identifier)?
("**to**" identifier)?

The aggregate operator (in *definition 3*) forms a new class from an existing set of attributes inside a single class. These properties are replaced with a relationship inside the original class. The operator $\triangleright$ denotes an aggregate operation, and the expression $\mathtt{B(b_0)} \triangleright \mathtt{A(a_0,a_1,..a_i)}$ states that a new class A is formed from an existing class B using properties a_0 to a_i, with a_0 and b_0 representing the newly formed relationship properties.

Definition 4: class_expand_dcl:
"**expand**" double_qualifier_dcl

The expand operator performs the reverse operation to the aggregate operator. This operation is achieved by expanding the named relationship property inside one class into the full set of properties. The operator $\triangleleft$ denotes an expand operation, and the expression $\mathtt{B(b_0)} \triangleleft \mathtt{A(a_0,a_1,..a_n)}$ states that the class A will disappear, and that properties a_1 to a_n are placed inside class B. As a side-effect of this operation, the relationship properties b_0 and a_0 will be hidden.

Definition 5: class_subclass_dcl:
identifier := "**subclass**" (identifier (, identifier)*)
"**from**" qualifier_dcl

The subclass operator creates a new subclass from an existing class by removing n properties from the existing class and placing them inside the newly defined subclass. A new *ISA* relationship is formed between the new subclass and the original class. The operator $\veebar$ indicates a subclass operation, and the expression $\mathtt{B} \veebar \mathtt{A(a_1,..,a_i)}$ states that a new class A is formed from properties a_1 to a_i, and the new class A is a subclass of B. As a side effect the properties a_1 to a_i are hidden from the class B.

2.2 Integration Operators

The view mechanism contains five integration operators: join, ojoin, superjoin, osuperjoin, and nulljoin. For space reasons, we will cover two of them in detail, together with rules for managing conflicts. A full description can be found in [14].

Definition 6: class_join_dcl:
"**join**" `qualifier_dcl`
"**from**" `qualifier_dcl, qualifier_dcl`
"**on**" `identifier`
`(link_dcl)*`
`(with_dcl)*`

The operator $\otimes$ indicates a `join` operation, and the expression `C←B(b0)` $\otimes$ `A(a0)` states that a new class *C* is formed from classes *A* and *B*, using the joining predicate $a_0{=}b_0$. The `join` operator merges two classes by placing all properties from the original classes inside a single virtual class. If the two classes have more than one attribute in common, then the view mechanism assumes that these will have equal values (i.e. they are compared in a pairwise fashion). If not, it will be either necessary (in the view specification) to rename one attribute, elect one value in preference to the other, or redefine one attribute so that both values are equal. Conflict resolution is discussed shortly.

Definition 7: class_superjoin_dcl:
"**superjoin**" `qualifier_dcl`
"**from**" `qualifier_dcl, qualifier_dcl`
"**on**" `identifier`
`(link_dcl)*`
`(with_dcl)*`

The `superjoin` operator is used to connect two classes through the common overlap of a subset of their attributes. The `superjoin` is a composite operator: it is a combination of the `join` and `superclass` primitive operators. The result is three classes: the new superclass containing common attributes, one new subclass containing the remaining attributes of the first source class, and a second new subclass containing the remaining attributes of the second source class. The operator $\circledast$ is used to indicate a `superjoin` operation, and the expression `C←B(b1,..,bi)` $\circledast$ `A(a1,..,ai)` states that a new class *C* is formed from classes *A* and *B* using *i* properties (b_1 to b_i) from *B* and (a_1 to a_i) from *A*. As a side-effect, both existing classes *A* and *B*, will lose these properties, although both will inherit them from the new superclass *C*. An outerjoin operation (`osuperjoin`) is also available.

Definition 8: link_dcl and *with_dcl:*
"**link**" `identifier := identifier`
"**with**" `( with_rename_rule | with_subclass_rule | with_prefer_rule| with_redefine_rule ) ;`

In *definition 8* the `link` and `with` declarators are shown. The `link` declarator is used to bind two attributes with different names so that they can be used as a joining predicate. The `with` declarator is used where two attributes may be used in the joining predicate, but for reasons of semantics or structural differences, have different local values.

- The `rename` rule can be used to rename one of the attributes, and thus, remove it form the joining expression.
- The `subclass` rule is used only with the `superjoin` operation, and is used again, to eliminate those attributes from the joining expression. This time, it is achieved by placing both attributes in opposite subclasses (after a `superjoin`).
- The `prefer` rule is used only for secondary joining attributes. For example, if *PatientID* is the joining property, but both classes also contain an *address* property in common, this would form part of the join (or superjoin) operation. By using the `prefer` rule to favour one attribute over the other, the `join` operation will unite both attributes, and where a conflict of addresses takes place, one *address* value is preferred over the other.
- The `redefine` rule is used to redefine a property where both schemata are known to employ different numerical formats. For example, if one database uses *Fahrenheit* and the other *Celsius*, a numerical expression can be applied to values of one schema before the `join` operation.

2.3 The ODL$_w$ Language

For completeness, a brief overview of the wrapper language is provided now.

Definition 9: The ODL$_w$ wrapper statement

```
"wrapper" identifier (RW)?
{ (class_dcl)+ }
```

A single `wrapper` statement is used to map one ODMG schema to a non-ODMG IS. Each statement map comprise multiple `class` declarations which represent the mapping of an ODMG class to some entity in the IS. A wrapper is marked as read-write (RW) only if it supports some form of locking, thus allowing update transactions.

Definition 10: class_dcl:

```
"class"{
     name_dcl
     (attribute_dcl)?
     (relationship_dcl)?}
```

A `class_dcl` production comprises a name declaration and any number of attribute and relationship declarations. The local entity may map to more than one ODMG class where inheritance is not used in the local IS. The remaining productions provide mapping names for entities and their properties and are described in depth in [14].

3 View Language Usage

In this section a sample integration process demonstrates the usage of the view language. Assume that two local ISs have undergone the local schema transformation stage, and now exist as ODMG schemata. Our implementation assumes that data always resides in the local system, and that the ODMG schemata act as wrappers, defined using the ODL$_W$ language. When defining views, it is necessary to understand the issue of generating extents for virtual classes, and this is based on *pivotal* classes and the fact that an object preserving semantics is employed.

Pivotal Classes. A view comprises one or more *schema segments*, where a segment is a subset of the base (or some underlying virtual) schema. When a view segment contains multiple classes, one class must be nominated as the *pivotal* class, which subsequently determines the extent for all classes in the view segment in which it is contained. A pivotal class may use a *filter* to reduce the extent of objects for each virtual class in the view segment. If the subschema contains multiple schema segments, each segment must have its own pivotal class and extent. If no extent is specified, then the extent of the base pivotal class(es) is used.

Object Preserving Semantics. No new identifiers are ever constructed as a result of a new virtual object. Where a virtual class is derived from a single base class, the base class identifier is used, and the onus is on the view mechanism to provide access to the virtual class (and not the base class which has the same oid). Every attribute and relationship property in the view schema is connected to a base class equivalent, thus facilitating the updating of view properties. There is one notable exception. Since an attribute may be a collection of literals, it is unclear how individual elements in such a collection can be updated since they contain no identifiers. This weakness in the ODMG model was highlighted in [19], and must be rectified in order to facilitate updates.

3.1 Schema Integration with ODL$_V$

This research is based on the integration of healthcare systems and examples are used to define views on a Patient Administration Systems (PAS) and a HIV System. Note that for global (or federated) views, there cannot be base classes, as the architecture (described in §4) assumes that the Federated Kernel has no database schema, and is used only for importing schemata from component systems, and the definition of federated schemata. In *example 1*, a local view is defined to retrieve the set of Patients having bloodtype *A*, and the Consultants to which they have links.

Example 1. Defining a simple class filter:

```
database PAS
subschema PasLocal5 {
      SchemaSegment{
            ImportBase Patient, Consultant;
            Restructure {
                use Patient
                class {
                    rename Patient as aPatient
                    filter aPatient where BloodType ='A' }
             };
        Export PasLocal5.aPatient, PasLocal5.Consultant; };
    };
```

The target database is named; this view (named *PasLocal5*) contains only one schema segment, which itself contains two classes of which the *Patient* class is the pivotal class.

In *example 2*, an example of an `aggregate` operation is shown. Some view expressions (database name and schema segment) have been omitted for space reasons.

Example 2. Defining an aggregate class:

```
    subschema AddressData {
        ImportBase Person, Patient;
        Restructure {
            use Patient
            class {
                Residence := aggregate (Street,City,PostCode)
                from Patient as resides to PatLink }
        };
    Export AddressData.Person, AddressData.Patient, AddressData.Residence; };
```

In *example 3* a join operation is used to merge two views: one from the PAS database and the second from the HIV database. The joining class is the *Patient* class and in this case it is assumed that *PatientID* (PAS) and *MRN* are identical (HIV); the *Quantity* field (used to indicate medication level) must be identical to join objects from both view extents, but *Quantity* is recorded in ounces in the PAS databases, and in grams in the HIV database. Finally, only Patients with bloodtype *O* are required in this view.

In *figure 1* both the local (imported) schemata, and the federated schema are illustrated, and in *example 3* the schema definition is presented.

Example 3. Defining the federated schema in figure 1:

```
    subschema Fed {
      ImportVirtual Pas.Patient, Pas.Person, Pas.Consultant, Pas.Drug, Hiv.Patient,
Hiv.Person, Hiv.Episodes;
        Restructure {
            use Pas.Patient, Hiv.Patient
            class {
                join Patient from Pas.Patient, Hiv.Patient on PatientId
                link PatientId := MRN;
                with Pas.Patient.Quantity = Quantity / 0.03527;
                filter Patient where bloodtype = 'O'; }
        };
        Export Fed.Person, Fed.Patient, Fed.Consultant, Fed.Drug, Fed.Episodes; };
```

An exhaustive collection of sample views containing all of the operators, defined at both local and global levels are provided in [14]. In the following section, a discussion on the generation of virtual class extents is presented.

3.2 Generating Virtual Class Extents

To generate the extents for virtual classes (using an object-preserving semantics), it is necessary to take the extent of the base class, and apply some predicate to generate the virtual extent. Where a class is a specialised class, an extent is generated for the specialised class, and all of its superclasses. Note that for the *Patient* instance in *figure 1*, its *Person* abstract instance will have the same 'unique' identifier: it is the same object, but in different roles.

Before discussing how *virtual* class extents are generated, it is necessary to understand how a similar method would generate *base* class extents. It is necessary at this point to introduce the concept of *shallow* and *deep* extents: a shallow extent (E_s) is the set of all objects constructed specifically for that class; and a deep extent (E_d) is the set of all objects and subclass objects constructed for a given class. The full (deep) extent of a class C with i subclasses S is given below.

General Expression for Base Class Extent:

$$E_d(C) = E_s(C) \cup E_d(S_1) \cup E_d(S_2) \cup ... \cup E_d(S_i)$$

This generalised expression states that the deep extent E_d for the base class C can be expressed as the shallow extent E_s for C plus the union of the deep

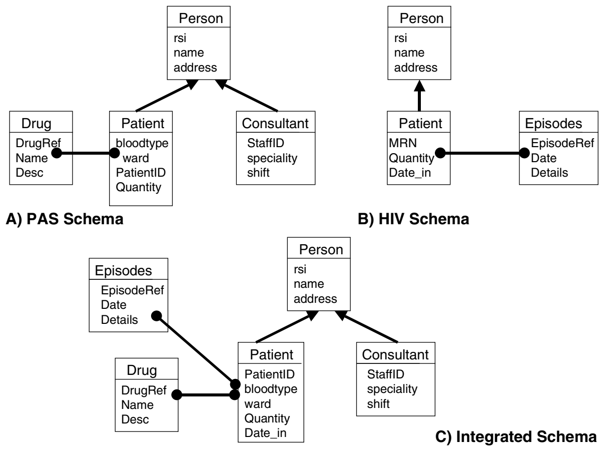


Fig. 1. Integration of PAS and HIV Schemata

extents (ΣE_d) of all of subclasses S of C. Furthermore, each deep extent E_d can be reduced to a set of shallow extents E_s through recursive procedures. Thus, in our perception, a class may have multiple extents, and each shallow extent can be generated by applying simple "select *" queries against each individual class.

In general, the generation of extents for virtual classes is similar to that of base class extents, except that a filter function `f()` must be applied to the class which affects the class extent[1]. Every class in the segment is directly or indirectly connected to the pivotal class and has their extent affected by the filter.

General Expression for Virtual Class Extent:

$$E_d(\ f(VC)\ ) = E_s(\ f(VC)\ ) \cup E_d(\ f(VS_1)\ ) \\ \cup E_d(\ f(VS_2)\ ) \cup \ldots \cup E_d(\ f(VS_i)\ )$$

However, unlike base class extents, the extent generating queries are applied to the classes to which these virtual classes are mapped. These may be base or virtual classes, and the mappings may be *one-to-one* or *one-to-many*, depending on the operation used to define the current virtual class. Thus, the shallow extent for virtual class *VC* is *the union of the extents of the classes to which it is mapped*, with each extent subsequently passed through the filter function `f()`.

The `join` operation in *example 3* results in a federated view comprising five classes, where two classes are join classes. The view mechanism executes a `join` operation on the *Patient* class (explicit in the view definition) and a subsequent (implicit) `join` operation on the superclass *Person* since both arguments (*Patient* classes) in the join operation have the same superclass. Where join classes have superclasses with the same name, they are eliminated from the view schema unless both are identical, or one is preferred over the other [14]. The deep extents for all five view classes are generated as follows:

$$\begin{gathered}
E_d(f(\text{Fed.Drug})) = E_s(\ f(\text{Fed.Drug})\ ) \\
E_d(f(\text{Fed.Episodes})) = E_s(\ f(\text{Fed.Episodes})\ ) \\
E_d(f(\text{Fed.Consultant})) = E_s(\ f(\text{Fed.Consultant})\ ) \\
E_d(f(\text{Fed.Patient})) = E_s(\ f(\text{Fed.Patient})\ ) \\
E_d(f(\text{Fed.Person})) = E_s(\ f(\text{Fed.Person})\ ) \cup E_d(\ f(\text{Fed.Patient})\ ) \cup E_d(\\
f(\text{Fed.Consultant})\ )
\end{gathered}$$

The shallow extents for view classes are taken as the extents for those classes to which they are mapped. Thus for the *Person* and *Patient* view classes this is a binary mapping, and both will have multiple extents. Thus, the shallow extent for the *Patient* class will be redefined as:

$$E_s(\ f(\text{Fed.Patient})\ ) = E_s(\ f(\text{Pas.Patient}) \cup f(\text{Hiv.Patient})\ )$$

Now we apply the concept of any class having multiple extents to virtual classes, and use it to simplify the generation of extents. Once the extents have been generated, the view representation can be used to hide or restructure properties to adhere to the view definition.

[1] In base classes a filter function `f()` is also applied, but this will always be 'select *'.

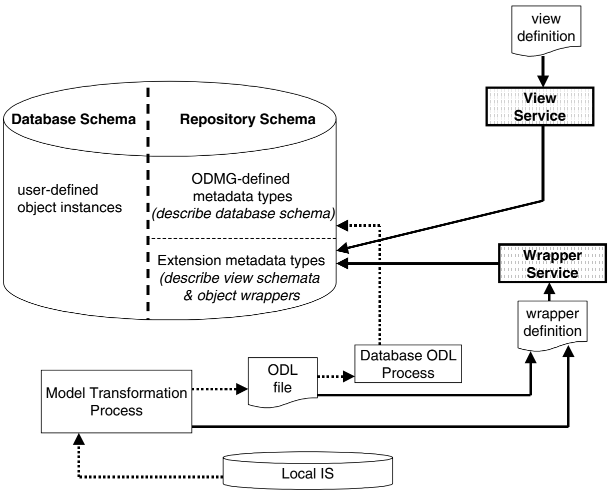

view definition
View Service
Database Schema
Repository Schema
user-defined object instances
ODMG-defined metadata types *(describe database schema)*
Extension metadata types *(describe view schemata & object wrappers*
Wrapper Service
wrapper definition
Database ODL Process
ODL file
Model Transformation Process
Local IS

Fig. 2. Service Architecture for local Information Systems

4 Architecture and Implementation

In this section we provide implementations details of the view architecture and services. The federated architecture used the ODMG model as a canonical model with the result that the component schema, its local views, the federated schema, and its global views, are all defined using the extended ODMG model and specification languages. The schema repository was extended to accommodate view metadata [12], and processors for both view and wrapper specifications were implemented using ANTLR [2], to define the productions, C++ to code the semantic actions for each production, and Versant [20] as the ODMG database implementation.

The process for integrating a new system requires a number of steps.

- An ODMG database is constructed with a schema representing the data to be shared with the federation. This hand-crafted step uses external research (such as [3][8]) to generate the component ODMG schema.
- A wrapper definition is specified using the ODL$_W$ language which maps entities and properties from the ODMG schema to the schema of the participating IS. The wrapper specification is passed through the *Wrapper Service* (see *figure 2*) which writes meta-objects to the (extended) schema repository of the ODMG database. The Object Manager can now extract data from the local IS when ODMG queries are generated.
- A different Object Manager is required for each data model. In our implementation, XML and ODBC object managers were constructed to build a federation of relational databases and XML data sources.
- Views are defined using the ODL$_V$ language, and passed through the *View Service* (see *figure 2*) which generates meta-objects as described in [14].

– An extraction process transfers views from participating systems (through their ODMG schema) to a federated kernel. This extraction process can employ one of four protocols: ODMG vendor-specific, ODMG generic, XML and CORBA protocols [13].
– At the federated kernel the ODLv language is again used to define views, although in this case, these specifications will use views imported from separate ISs.
– A *View Display Service* has been implemented to display views from local or global schemata.

Note that none of the three final steps are illustrated in *figure 2*. Outside the local operations shown in the illustration, is a *Federated Kernel*, which begins as an 'empty' ODMG database, used to import all of the locally defined views.

5 Conclusions and Future Research

Our research is focused on the creation of federations of healthcare systems. Traditionally, these systems have existed as large legacy systems often build using unlikely combinations of technologies (one such system was a COBOL application running on an old Unix platform). Recently, ODBC and XML interfaces have been built as wrappers to many of these systems, illustrating the fact that users now demand easy-to-use access methods, built using standard technologies. Our federated architecture uses a combination of different standards (ODMG and CORBA) to provide the federated middleware to facilitate the integration of these, still heterogenous, information sources. These systems required complex view mechanisms in order to facilitate the needs of global (or federated) schema construction. Our motivation to build our own view mechanism (view languages and services) was based on the fact that older systems employed proprietary models and software tools in federated schema construction. As a result, we specified a view layer for ODMG databases which allows both the creation of ODMG wrappers to multiple Information System types, and the construction of both local and global (semantically-rich) views. To demonstrate the view mechanism's usability, we implemented a prototype which contains all of the functionality required to construct federated schemata. The only limitation is that the Object Manager which resides between the ODMG Component Schema and the local IS can interact with ODBC databases and XML data stores only. Further Object Manager implementations are required for alternative ISs.

Current research is twofold: the incorporation of behaviour in ODMG views, and 'safe' update views. The former uses a combination of ODMG database technology, the ODLv view language, and distributed object technology to make class methods available at the Federated Kernel. The latter looks at the wrapper definitions, and then at combinations of view definitions to determine which federated views can allow updates safely.

References

1. Abiteboul S. On Views and XML. *SIGMOD Record* 28:4, pp. 30-38, ACM Press, 1999. 151
2. ANTLR Reference Manual. *http://www.antlr.org/doc/* 1999. 160
3. Batini C., Lenzerini M. and Navathe S. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18:4, December 1986. 160
4. Bukhres O. and Elmagarmid A. (eds.), *Object-Oriented Multidatabase Systems*, Prentice Hall, 1996. 151
5. Cattel R. et. al. (eds.) (2000). *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann.
6. Cattell R. and Barry D. (eds), *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann, 1997. 150
7. Dos Santos C., Abiteboul S. amd Delobel C. Virtual schemas and bases. *Advances in Database Technology* (EDBT94), pp. 81-94, Springer, 1994. 151
8. Fahrner C. and Vossen G. Transforming Relational Database schemata into Object-Oriented schemata According to ODMG-93. *Proceedings of 4th International Conference on Dedictive and Object-Oriented Databases (DOOD 95)*, pp 429-446, LNCS 1013, 1995. 160
9. Jordan D. *C++ Object Databases: Programming with the ODMG Standard*. Addison Wesley, 1998.
10. Motro A. Superviews: Visual Integration of Multiple Databases. *IEEE Transactions on Software Engineering*, 13:7, 1987. 151
11. Roantree M., Murphy J. and Hasselbring W. The OASIS Multidatabase Prototype. *ACM Sigmod Record*, 28:1, March 1999. 151
12. Roantree M., Kennedy J., and Barclay P. Using a Metadata Software Layer in Information Systems Integration. *Proceedings of 13th Conference on Advanced Information Systems Engineering (CAiSE 2001)*, pp. 299-314, LNCS 2068, June 2001. 160
13. Roantree M., Kennedy J., and Barclay P. Interoperable Services for Federations of Database Systems. To appear in *5th East-European Conference on Advances in Databases and Information Systems* (ADBIS 2001), Vilnius, September 2001. 161
14. Roantree M. Constructing View Schemata Using an Extended Object Definition Language. *PhD Thesis*. Napier University, November 2000. 152, 153, 155, 158, 159, 160
15. Rundensteiner E. Multiview: A Methodology for Supporting Multiple Views in Object-Oriented Databases. *Proceedings on the 18th International Conference on Very Large Databases (VLDB'92)*, pp 187-198, 1992. 151
16. Saltor F., Castellanos M. and Garcia-Solaco M. Suitability of Data models as Canonical Models for Federated Databases. *ACM SIGMOD Record,* 20:4, 1991. 151
17. Sheth A and Larson J. Federated Database Systems for Managing Distributed, heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22:3, pp 183-236, ACM Press, 1990. 150
18. Scholl M., Schek H. and Tresch M. Object Algebra and Views for Multi-Objectbases. In *Distributed Object Management*, Özsu, Dayel & Valdiurez (eds), pp. 353-374, Morgan Kaufmann, 1994. 151
19. Subieta K. Object-Oriented Standards: Can ODMG OQL be extended to a Programming Language? *Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications*, pp. 546-555, Japan, 1996. 156

20. Versant Corporation. *Versant C++ Reference Manual 5.2*, April 1999. 160

Deriving “Sub-source” Similarities from Heterogeneous, Semi-structured Information Sources

Domenico Rosaci, Giorgio Terracina, and Domenico Ursino

Dipartimento di Informatica, Elettronica, Matematica e Trasporti
Università degli Studi “Mediterranea” di Reggio Calabria
Via Graziella, Località Feo di Vito, 89100 Reggio Calabria, Italy
{rosaci,terracina,ursino}@ing.unirc.it

Abstract. In this paper we propose a semi-automatic technique for deriving the similarity degree between two portions of heterogeneous, semi-structured information sources (hereafter, sub-sources). The proposed technique consists of two phases: the first one selects the most promising pairs of sub-sources, whereas the second one computes the similarity degree relative to each promising pair. In addition, we show that the detection of sub-source similarities is a special case (and a very interesting one, for semi-structured information sources) of the more general problem of Scheme Match. Finally we discuss some possible applications which can benefit of derived sub-source similarities. A real example case is presented for better clarifying the proposed technique.

1 Introduction

Scheme Match, i.e. the activity of finding a mapping between those elements of two schemes that semantically correspond to each other, is becoming a challenging issue in the field of Information Systems. Indeed, it plays a central role in various applications, such as information source integration, e-commerce, scheme evaluation and migration, data and web warehousing, information source design and so on [6,10].

In the past, most of the proposed approaches for Scheme Match were manual; nowadays, the need of semi-automatic techniques for carrying out this task is widely recognized and, in fact, the most recent proposals in this context are semi-automatic.

Most of the techniques for Scheme Match proposed in the literature have been designed only for databases, i.e. structured information sources [2,4,5,13,8]. Such techniques aim at deriving terminological and structural relationships between *single classes of objects* (e.g., two entities, two relationships, an entity and a relationship, and so on).

Nowadays, due to the enormous spreading of the Web, the number of semi-structured information sources is exponentially increasing. As a consequence, new approaches to Scheme Match, handling semi-structured information sources,

© Springer-Verlag Berlin Heidelberg 2001

appear to be compulsory. Such approaches must be somehow different from the traditional approaches since, in semi-structured information sources, a concept is not generally expressed by a single class of objects but it is represented by a group of them; as an example, in XML, concepts are expressed by elements which can be, in their turn, represented by sub-elements. Moreover, different instances of the same element could have different structures. In such a situation, the emphasis shifts away from the extraction of semantic *correspondences between object classes* to the derivation of semantic *correspondences between groups of object classes*, each of which being, in practice, a little portion of information source (i.e., a little sub-source). Generalizing this line of reasoning, it is also interesting to analyze semantic correspondences holding amongst larger portions of information sources.

In this paper we propose a semi-automatic technique for extracting similarities between sub-sources belonging to different, heterogeneous and semi-structured information sources. Since these last can have different formats (such as OEM-Graphs, XML documents and so on), the adoption of a conceptual model, capable to uniformly handle information sources of different nature, appears to be extremely useful. Obviously, translation rules should exist from classical information source formats to the adopted conceptual model. In this paper we use the SDR-Network conceptual model [14], which satisfies the constraints described above. In addition, the SDR-Network has the important characteristic that a suitable metrics can be defined on it for measuring the strength of the semantic relationships existing between object classes belonging to the same information source; this metrics plays a central role in the technique for deriving the similarity degree between sub-sources. An overview of both the SDR-Network and the related metrics is presented in Section 2.

Given an information source IS, the number of possible sub-sources that can be derived from it is extremely high. In order to avoid handling huge numbers of sub-source pairs, we propose an heuristic technique for singling out only the most promising ones. A pair of sub-sources is considered "promising" if the sub-sources at hand include a large number of pairs of object classes whose similarity has been already established. In this way it is probable that the overall similarity degree, resulting for the promising sub-source pair, will be high. This heuristic technique is described in Section 3.

After that the most promising pairs of sub-sources have been selected, their similarity degree must be computed. The similarity degree associated to each pair of sub-sources is determined by computing the objective function associated to a maximum weight matching, defined on a suitable bipartite graph, constructed from the object classes composing the sub-sources of the pair. The idea underlying the adoption of weighted matching algorithms as the core step for "measuring" the similarity of two sub-sources SS_i and SS_j is as follows. SS_i can be detected to be similar to SS_j only if it is possible to single out object classes of SS_i and SS_j that are pairwise similar in their turn. The maximum weight matching algorithm is thus used to single out such matching portions. In

Section 4 we describe details relative to the computation of the similarity degree of two sub-sources.

In order to improve the comprehension of the proposed technique, a leading example is presented throughout the paper.

It is worth pointing out that the SDR-Network and its metrics have been already exploited in [9] for defining a technique for deriving synonymies and homonymies between object classes belonging to information sources having different formats and structures. Therefore, in the whole, we propose a *unified*, semi-automatic approach for deriving object class synonymies and homonymies, as well as sub-source similarities, relative to both structured and semi-structured information sources. In our opinion, this is particularly interesting since: *(i)* we are proposing the derivation of a property (i.e., sub-source similarity) which, generally, is not handled by most of the other approaches for Scheme Match proposed in the literature (which aim at deriving similarities and dissimilarities between *single* classes of objects); *(ii)* the technique proposed here is part of a more general framework whose purpose is to *uniformly* derive various kinds of terminological and structural properties (namely object class synonymies and homonymies as well as sub-source similarities).

In our technique the relative topology of the considered SDR-Networks influences the set of returned results. This fact makes it somehow related to the well known graph isomorphism problem, even if we do not use such a correspondence within our technique. Indeed, whereas graph isomorphism exclusively deals with the structural similitude between graphs, our technique heavily depends on the use of factors describing the semantics of the involved data sources which have no correspondence in the graph isomorphism setting.

Sub-source similarities we derive in this paper can be exploited in several contexts. Indeed, all applications of Scheme Match relative to synonymies between single classes of objects, considered in the context of databases (and described in detail in [10]), can be extended to similarities between sub-sources, considered in the context of semi-structured information sources. In more detail, sub-source similarities can be exploited for: *(i)* Information Source Integration [12], *(ii)* E-commerce, *(iii)* Semantic Query Processing [11,15], and, *(iv)* Data and Web Warehouses [1,10].
The next sections describe the proposed technique in detail.

2 The SDR-Network

Given an information source IS, the associated SDR-Network $Net(IS)$ is a rooted labeled graph [14]:

$$Net(IS) = \langle NS(IS), AS(IS) \rangle = \langle NS_A(IS) \cup NS_C(IS), AS(IS) \rangle$$

Here, $NS(IS)$ is a set of nodes, each one representing a class of objects (i.e., a concept) of IS. Each node is identified by the name of the class it represents. Nodes in $NS(IS)$ are subdivided in two subsets, namely the set of *atomic nodes* $NS_A(IS)$ and the set of *complex nodes* $NS_C(IS)$. A node is atomic if it does not have outgoing arcs, complex otherwise. Since an SDR-Network node represents a class of objects, from now on, we use the terms “SDR-Network node”, “object class” and “concept” interchangeably.

$AS(IS)$ denotes a set of arcs; an arc represents a relationship between two concepts. More specifically, an arc from S to T, labeled L_{ST} and denoted by $\langle S, T, L_{ST} \rangle$, indicates that the concept represented by S is semantically related to the concept denoted by T. S is called the “source node”, whereas T is the “target node” of the arc.

L_{ST} is a pair $[d_{ST}, r_{ST}]$, where both d_{ST} and r_{ST} belong to the real interval $[0, 1]$. d_{ST} is the *semantic distance coefficient*; it indicates how much the concept expressed by T is semantically close to the concept expressed by S; this depends from the capability of the concept associated to T to characterize the concept associated to S. As an example, in an E/R scheme, an attribute A is semantically closer to its entity E than another entity E_1 related to E by a relationship R; analogously, in an XML document, a sub-element E_1 of an element E is closer to E than another element E_2 which E refers by an $IDREF$ attribute. r_{ST} is the *semantic relevance coefficient*, representing the fraction of instances of the concept denoted by S whose complete definition requires at least one instance of the concept denoted by T.

An example of an SDR-Network relative to an information source handling European Social Funds (hereafter ESF) is shown in Figure 1. In the figure, in order to simplify its layout, a grey node having name x is used to indicate that the arc incident onto x must be considered incident onto the corresponding white node having the same name. SDR-Network nodes such as *Operative Program*, *Project*, *Ministry*, etc. represent the corresponding object classes. The arc $\langle$*Operative Program*, *Ministry*, $[1, 0.6]\rangle$ denotes the existence of a relationship between *Operative Program* and *Ministry*; in particular, it indicates that 60% of operative programs are managed by a ministry. The other arcs have an analogous semantics.

Note that, basically, any information source can be represented as a set of concepts and a set of relationships among them. Since SDR-Network nodes and arcs are well suited to represent such concepts and relationships, the SDR-Network can be used to uniformly model most existing information sources. In this respect, semantic preserving translations have been provided from some interesting source formats, such as XML, OEM and E/R, to SDR-Networks [7,9,14].

The SDR-Network has the important characteristic that a suitable metrics can be defined on it for measuring the strength of the semantic relationships existing between object classes belonging to the same information source; this metrics plays a central role in the technique for deriving the similarity degree between sub-sources we propose in this paper. The metrics is based on the following definitions:

Definition 1. The *Path Semantic Distance* of a path P in $Net(IS)$ (denoted by PSD_P) is the sum of the semantic distance coefficients associated to the arcs constituting the path.

The *Path Semantic Relevance* of a path P in $Net(IS)$ (denoted by PSR_P) is the product of the semantic relevance coefficients associated to the arcs constituting the path.

A D_Path_n is a path P in $Net(IS)$ such that $n \leq PSD_P < n+1$.

The *CD_Shortest_Path* (Conditional D_Shortest_Path) between two nodes N and N' in $Net(IS)$ and including an arc A (denoted by $\lfloor N, N' \rfloor_A$) is the path having the minimum Path Semantic Distance among those connecting N and N' and including A. If more than one path exists having the same minimum Path Semantic Distance, one of those having the maximum Path Semantic Relevance is chosen. $\Box$

Definition 2. Define the *i_th neighborhood* of a node $x \in Net(IS)$ as:
$nbh(x,i) = \{A | A \in AS(IS), A = \langle z, y, l_{zy} \rangle, \lfloor x, y \rfloor_A \; is \; a \; D\_Path_i, x \neq y\} \quad i \geq 0$ $\Box$

Thus, an arc $\langle z, y, l_{zy} \rangle$ belongs to $nbh(x,i)$ if there exists a CD_Shortest_Path from x to y, including $\langle z, y, l_{zy} \rangle$, which is a D_Path_i; note that, as such, $A \notin nbh(x,j), j < i$. Finally, it is worth pointing out that x may coincide with z.

3 Selection of Promising Pairs of Sub-sources

3.1 Overview

Consider an information source IS and the corresponding SDR-Network $Net(IS)$; the number of possible sub-sources that can be identified in IS is exponential in the number of nodes of $Net(IS)$. To avoid the burden of analyzing such a huge number of sub-sources, we have defined a technique for singling out the most *promising* ones. The proposed technique receives two information sources IS_1 and IS_2 (represented by the corresponding SDR-Networks $Net(IS_1)$ and $Net(IS_2)$) and a Dictionary SD of Synonymies between nodes of $Net(IS_1)$ and $Net(IS_2)$. SD can be obtained by applying the technique presented in [9]. Synonymies are represented in SD by tuples of the form $\langle N_i, N_j, f_{ij} \rangle$, where N_i and N_j are the synonym nodes and f_{ij} is a coefficient in the real interval $[0,1]$, indicating the similarity degree of N_i and N_j. The technique derives the most promising pairs of sub-sources according to the following rules:

- It considers those pairs of sub-sources $[SS_i, SS_j]$ such that $SS_i \in Net(IS_1)$ is a rooted sub-net having a node N_i as root, $SS_j \in Net(IS_2)$ is a rooted sub-net having a node N_j as root and N_i and N_j are synonyms[1].
- In order to select only the most promising pairs of sub-sources, having the synonym nodes N_i and N_j as roots, the technique computes the maximum weight matching on some suitable bipartite graphs obtained from the target nodes of the arcs forming the neighborhoods of N_i and N_j.
- Given a pair of synonym nodes N_i and N_j, the technique derives a *promising pair of sub-sources* $[SS_{i_k}, SS_{j_k}]$, for each k such that both $nbh(N_i, k)$ and $nbh(N_j, k)$ are not empty. SS_{i_k} and SS_{j_k} are constructed by determining

[1] Note that N_i and N_j must be complex nodes, since atomic nodes do not have outgoing arcs and, therefore, cannot be roots.

the *promising pairs of arcs* $[A_{i_k}, A_{j_k}]$ such that $A_{i_k} \in nbh(N_i, l), A_{j_k} \in nbh(N_j, l)$, for each l belonging to the integer interval $[0, k]$.
A *pair of arcs* $[A_{i_k}, A_{j_k}]$ is considered *promising* if *(i)* an edge between the target nodes T_{i_k} of A_{ik} and T_{j_k} of A_{jk} is present in the maximum weight matching computed on a suitable bipartite graph constructed from the target nodes of the arcs of $nbh(N_i, l)$ and $nbh(N_j, l)$, for some l belonging to the integer interval $[0, k]$; *(ii)* the similarity degree of T_{i_k} and T_{j_k} is greater than a certain given threshold.

The rationale underlying this approach is that of constructing promising pairs of sub-sources such that each pair is composed by the maximum possible number of pairs of object classes whose synonymy has been already stated. In this way it is probable that the overall similarity degree, resulting for each promising pair of sub-sources, will be high.

3.2 Technical Details

Here we formalize our technique for selecting the most promising pairs of sub-sources. From now on, since each source is represented by the corresponding SDR-Network, we use the terms “sub-source” and “sub-net” interchangeably; in addition we represent a sub-net with the set of arcs composing it.

The set SPS of the most promising pairs of sub-sources, associated to two information sources IS_1 and IS_2, represented by the corresponding SDR-Networks SDR_1 and SDR_2, is obtained as follows:

$$SPS = \sigma(SDR_1, SDR_2, SD, \pi(SD))$$

Here, the function π takes a Synonymy Dictionary SD as its input and returns the set ISP of the most interesting pairs of synonym nodes. π constructs ISP by selecting those triplets $\langle N_i, N_j, f_{ij} \rangle$ of SD such that N_i and N_j are both complex nodes and f_{ij} is greater than a certain, dynamically computed, threshold th_{Syn}. This is obtained as $th_{Syn}(T) = max\left(f_{max} \times \theta_{Syn}, th_m^{Syn}\right)$, where *(i)* f_{max} represents the maximum value of the coefficients associated to the tuples of T; *(ii)* θ_{Syn} is a tuning coefficient belonging to the real interval $[0, 1]$; *(iii)* th_m^{Syn} is a minimum acceptable value for the similarity coefficient. We have experimentally set $\theta_{Syn} = 0.65$ and $th_m^{Syn} = 0.50$.

A node N_i can be synonym of only one node of the other SDR-Network. If more than one triplet exists, involving a node N_i and having a coefficient greater than th_{Syn}, π selects only one of them; this task is carried out by computing a maximum weight matching on the bipartite graph constructed from the set of nodes of the two SDR-Networks and having an edge between two nodes N_i and N_j only if the similarity degree between N_i and N_j is greater than th_{Syn}.

The function σ receives two SDR-Networks SDR_1 and SDR_2, a Synonymy Dictionary SD and a set ISP of interesting synonym pairs. For each tuple $\langle N_i, N_j, f_{ij} \rangle \in ISP$, σ calls a function β for deriving the set of the most promising pairs of sub-nets relative to N_i and N_j. σ is defined as:

$$\sigma(SDR_1, SDR_2, SD, ISP) = \bigcup_{\langle N_i, N_j, f_{ij}\rangle \in ISP} \beta(SDR_1, SDR_2, SD, N_i, N_j)$$

The function β activates, for each k such that both $nbh(N_i, k)$ and $nbh(N_j, k)$ are not empty, a function β_k for determining the promising pair of sub-nets relative to N_i, N_j and k. Since the sub-nets composing the pair returned by β_k could contain arcs not connected to N_i and N_j, and since we are interested only in rooted connected sub-nets, having N_i and N_j as roots, a function ν must be applied on the output of β_k for discarding unconnected arcs. The function β is encoded as follows:

$$\begin{aligned}\beta(SDR_1, SDR_2, SD, N_i, N_j) = \{\nu(\beta_k(SDR_1, SDR_2, SD, N_i, N_j), N_i, N_j) \mid \\ nbh(N_i, k) \neq \emptyset,\ nbh(N_j, k) \neq \emptyset\}\end{aligned}$$

The function β_k returns a *promising pair of sub-nets* relative to N_i, N_j and k. In particular, if $k = 0$, β_k returns the two sub-nets composed by the set of *promising arcs* relative to $nbh(N_i, 0)$ and $nbh(N_j, 0)$. If $k > 0$, β_k returns the two sub-nets obtained by composing the pair of sub-nets constructed by β_{k-1} with the pair of sets of promising arcs relative to $nbh(N_i, k)$ and $nbh(N_j, k)$.

$$\beta_k(SDR_1, SDR_2, SD, N_i, N_j) = \begin{cases} \mu(SD, N_i, N_j, 0) & \text{if } k = 0 \\ \beta_{k-1}(SDR_1, SDR_2, SD, N_i, N_j) \uplus \mu(SD, N_i, N_j, k) & \text{if } k > 0 \end{cases}$$

Given two pairs of sets $P' = [AS'_1, AS'_2]$ and $P'' = [AS''_1, AS''_2]$, $P' \uplus P''$ indicates the pair $[AS'_1 \cup AS''_1, AS'_2 \cup AS''_2]$.

The function μ receives a Synonymy Dictionary SD, two nodes N_i and N_j and an integer k and selects the pair of sets of promising arcs $[AS_{i_k}, AS_{j_k}]$ relative to $nbh(N_i, k)$ and $nbh(N_j, k)$. $[AS_{i_k}, AS_{j_k}]$ is constructed by taking those pairs of arcs $[A_{l_k}, A_{m_k}]$ such that:

- $A_{l_k} = \langle S_{l_k}, T_{l_k}, L_{l_k}\rangle$ belongs to $nbh(N_i, k)$ and $A_{m_k} = \langle S_{m_k}, T_{m_k}, L_{m_k}\rangle$ belongs to $nbh(N_j, k)$;
- The edge $\langle T_{l_k}, T_{m_k}, [s_{lm}, r_{lm}]\rangle$ is present in the maximum weight matching on a suitable bipartite graph obtained from the target nodes of the arcs composing $nbh(N_i, k)$ and $nbh(N_j, k)$.
- The similarity degree s_{lm} associated to T_{l_k} and T_{m_k} is greater than a certain given threshold th_μ (we have empirically set $th_\mu = 0.25$).

More formally the function μ is as follows:

$$\begin{aligned}\mu(SD, N_i, N_j, k) = \biguplus\{[\{A_{l_k}\}, \{A_{m_k}\}] \mid A_{l_k} \in nbh(N_i, k), A_{m_k} \in nbh(N_j, k), \\ A_{l_k} = \langle S_{l_k}, T_{l_k}, [d_{l_k}, r_{l_k}]\rangle, A_{m_k} = \langle S_{m_k}, T_{m_k}, [d_{m_k}, r_{m_k}]\rangle, \\ \langle T_{l_k}, T_{m_k}, [s_{lm}, r_{lm}]\rangle \in \vartheta(SD, N_i, N_j, k), s_{lm} > th_\mu\}\end{aligned}$$

Here, the function ϑ returns the set of edges composing the maximum weight matching on the bipartite graph obtained from the target nodes of the arcs belonging to $nbh(N_i, k)$ and $nbh(N_j, k)$. ϑ is encoded as follows:

$$\vartheta(SD, N_i, N_j, k) = \psi(SD, \rho(\tau(N_i, k), \tau(N_j, k)))$$

The function $\tau(N, k)$ takes a node N and an integer k as input and returns a set of pairs (N_l, r_l) such that N_l is the target node of an arc belonging to $nbh(N, k)$ and r_l represents the semantic relevance of N_l w.r.t. N. In particular:

$$\tau(N, k) = \{(N_l, r_l) \mid \langle N_m, N_l, [d_{ml}, r_{ml}]\rangle \in nbh(N, k), r_l = \varrho(\lfloor N, N_l \rfloor_{\langle N_m, N_l, [d_{ml}, r_{ml}]\rangle})\}$$

Recall that the notation $\lfloor N, N_l \rfloor_A$ indicates the CD-Shortest Path from N to N_l including the arc A (see Definition 1); the function $\varrho(P)$ takes in input a path P and computes the Path Semantic Relevance of P (see Definition 1).

The function $\rho(PS_i, PS_j)$ receives two sets of pairs $PS_i = \{(N_{i_1}, r_{i_1}), \ldots, (N_{i_p}, r_{i_p})\}$ and $PS_j = \{(N_{j_1}, r_{j_1}), \ldots, (N_{j_q}, r_{j_q})\}$. ρ modifies either PS_i or PS_j in such a way that they have the same cardinality; this task is carried out by adding some fictitious pairs to the set having the lesser number of pairs. Obviously names assigned to fictitious nodes are not significant so that they are not involved in any synonymy stored in SD.

The function $\psi(T, PS_i, PS_j)$ computes the maximum weight matching on a particular bipartite graph, as explained next. The inputs here are a set T of triplets denoting similarities between nodes, and two sets of pairs $PS_i = \{(N_{i_1}, r_{i_1}), \ldots, (N_{i_p}, r_{i_p})\}$ and $PS_j = \{(N_{j_1}, r_{j_1}), \ldots, (N_{j_p}, r_{j_p})\}$, as returned by the function ρ. The output is the set of edges composing the matching.

First ψ constructs a bipartite graph $BG = (U \cup V, E)$ where:

- U is a set of nodes; the node $u_l \in U$ corresponds to the node N_{i_l} of a pair $(N_{i_l}, r_{i_l}) \in PS_i$;
- V is a set of nodes; the node $v_m \in V$ corresponds to the node N_{j_m} of a pair $(N_{j_m}, r_{j_m}) \in PS_j$;
- E is a set of edges; in particular, there is an edge for each pair (u_l, v_m), $u_l \in U$ and $v_m \in V$. Each edge has a label $L_{lm} = [s_{lm}, r_{lm}]$; here $s_{lm} = f$ if $\langle u_l, v_m, f\rangle \in T$, $s_{lm} = 0$ otherwise; $r_{lm} = 0.5 \times r_{i_l} + 0.5 \times r_{j_m}$.

The maximum weight matching for BG is defined, in this case, as a set $E' \subseteq E$ of edges such that, for each node $x \in (U \cup V)$, there is exactly one edge of E' incident onto x and $\phi(E') = \frac{\sum_{\langle u_l, v_m, [s_{lm}, r_{lm}]\rangle \in E'} (s_{lm} \times r_{lm})}{\sum_{\langle u_l, v_m, [s_{lm}, r_{lm}]\rangle \in E'} r_{lm}}$ is maximum; we assume that if $\sum_{\langle u_l, v_m, [s_{lm}, r_{lm}]\rangle \in E'} r_{lm} = 0$ then $\phi(E') = 0$ (for algorithms deriving the maximum weight matching of a bipartite graph see [3]). The output of ψ, i.e. the maximum weight matching, is exactly E'.

Finally, the function ν takes in input a pair of sets of arcs $[AS_{i_k}, AS_{j_k}]$, as returned by the function β_k, and two nodes N_i and N_j and, for each set of the pair, calls a function ν' which filters out those arcs of the set not connected, through the other arcs of the set, to N_i and N_j, resp. The function ν is as follows:

$$\nu([AS_{i_k}, AS_{j_k}], N_i, N_j) = [\nu'(AS_{i_k}, N_i), \nu'(AS_{j_k}, N_j)]$$

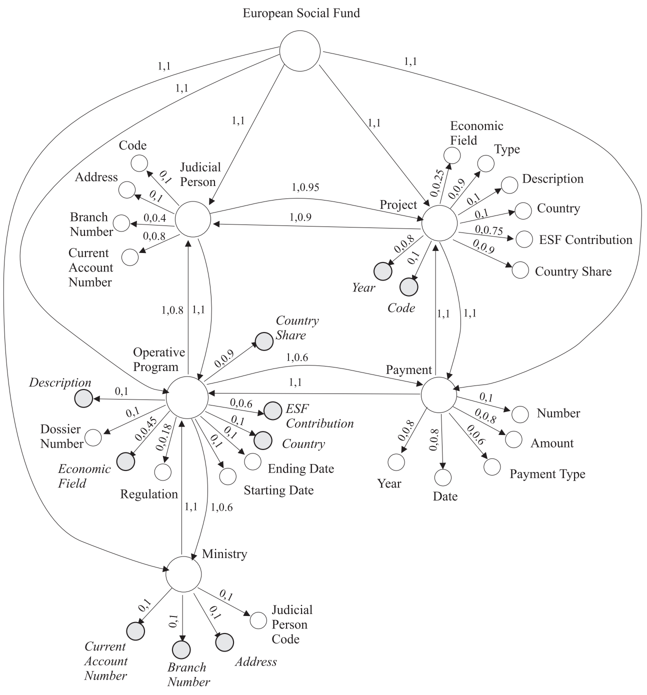


Fig. 1. The SDR-Network of the ESF information source

3.3 A Real Example Case

Here we show the behaviour of the proposed technique on a real example case. In particular, we consider two information sources relative to European Social Funds (hereafter ESF) and to European Community Projects (hereafter ECP). These are derived from the corresponding ones owned by Italian Central Government Offices. Due to space constraints, in Figures 1 and 2, we show only the corresponding SDR-Networks.

By applying the technique for deriving synonymies and homonymies between object classes presented in [9], we obtain the Synonymy Dictionary SD shown in Table 1[2].

The set SPS of the most promising pairs of sub-sources relative to ESF and ECP is constructed by calling the function σ as follows:

$$SPS = \sigma(ESF, ECP, SD, \pi(SD))$$

π determines the set ISP of the most interesting pairs of synonym nodes. As for our application case, $th_{Syn} = max(0.65 \times 0.65, 0.50) = 0.50$ and, therefore,[3]

[2] Due to space constraints, the table shows only similarities between complex nodes.

[3] Here and in the following we use the notation $N_{[S]}$ for indicating the node N of the SDR-Network S

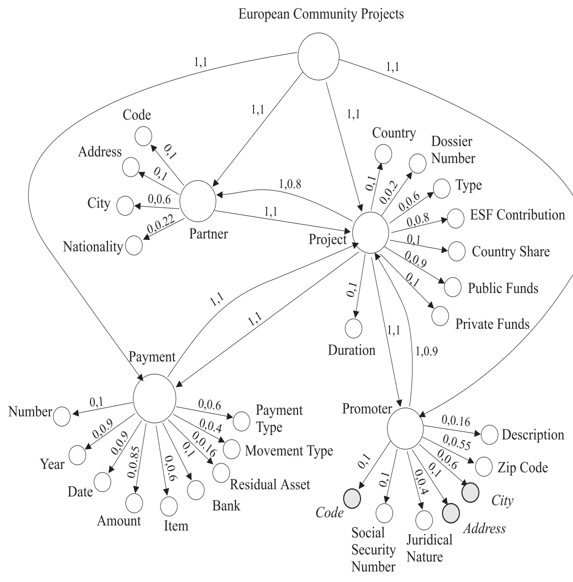


Fig. 2. The SDR-Network of the ECP information source

Table 1. Similarities between complex nodes of ESF and ECP

First Object	*Second Object*	*Sim*
Judicial Person	Partner	0.59
Judicial Person	Payment	0.30
Judicial Person	ECP	0.04
Project	Partner	0.14
Project	Payment	0.18
Operative Program	Partner	0.13
Operative Program	Payment	0.16
Operative Program	ECP	0.04
Payment	Partner	0.03
Payment	Payment	0.65
Ministry	Partner	0.32
Ministry	Payment	0.17
Ministry	ECP	0.04
ESF	Partner	0.06
ESF	Payment	0.04
Judicial Person	Project	0.05
Judicial Person	Promoter	0.47
Project	ECP	0.02
Project	Project	0.63
Project	Promoter	0.14
Operative Program	Project	0.51
Operative Program	Promoter	0.02
Payment	ECP	0.07
Payment	Project	0.06
Payment	Promoter	0.25
Ministry	Project	0.04
Ministry	Promoter	0.22
ESF	ECP	0.30
ESF	Project	0.03
ESF	Promoter	0.06

$$ISP = \{\langle Judicial\ Person_{[ESF]}, Partner_{[ECP]}, 0.59\rangle,$$
$$\langle Payment_{[ESF]}, Payment_{[ECP]}, 0.65\rangle, \langle Project_{[ESF]}, Project_{[ECP]}, 0.63\rangle\}$$

Note that the tuple $\langle Operative\ Program_{[ESF]}, Project_{[ECP]}, 0.51\rangle$ does not belong to ISP since it comprises also a similarity between $Project_{[ESF]}$ and $Project_{[ECP]}$ characterized by a higher similarity degree.

σ calls the function β for each tuple of ISP; in our application case we have:

$$\sigma(ESF, ECP, SD, IPS) =$$
$$\beta(ESF, ECP, SD, Judicial\ Person_{[ESF]}, Partner_{[ECP]}) \cup$$
$$\beta(ESF, ECP, SD, Payment_{[ESF]}, Payment_{[ECP]}) \cup$$
$$\beta(ESF, ECP, SD, Project_{[ESF]}, Project_{[ECP]})$$

In the following, due to space constraints, we show only the behaviour of

$$\beta(ESF, ECP, SD, Project_{[ESF]}, Project_{[ECP]})$$

Since $nbh(Project_{[ESF]}, k)$ and $nbh(Project_{[ECP]}, k)$ are both not empty only for $k = 0$, $k = 1$ and $k = 2$, we have:

$$\begin{aligned}
&\beta(ESF, ECP, SD, Project_{[ESF]}, Project_{[ECP]}) = \{\\
&\ \nu(\beta_0(ESF, ECP, SD, Project_{[ESF]}, Project_{[ECP]}), Project_{[ESF]}, Project_{[ECP]}),\\
&\ \nu(\beta_1(ESF, ECP, SD, Project_{[ESF]}, Project_{[ECP]}), Project_{[ESF]}, Project_{[ECP]}),\\
&\ \nu(\beta_2(ESF, ECP, SD, Project_{[ESF]}, Project_{[ECP]}), Project_{[ESF]}, Project_{[ECP]})\}
\end{aligned}$$

In order to provide an example of the behaviour of β_k, we illustrate the tasks carried out by β_0. Recall that:

$$\begin{aligned}
&\beta_0(ESF, ECP, SD, Project_{[ESF]}, Project_{[ECP]}) =\\
&\qquad\qquad\qquad\qquad \mu(SD, Project_{[ESF]}, Project_{[ECP]}, 0)
\end{aligned}$$

The function μ computes a maximum weight matching on a suitable bipartite graph, obtained from the target nodes of the arcs composing $nbh(Project_{[ESF]}, 0)$ and $nbh(Project_{[ECP]}, 0)$. In particular,

$$\begin{aligned}
&nbh(Project_{[ESF]}, 0) = \{\langle Project_{[ESF]}, Year_{[ESF]}, [0, 0.8]\rangle,\\
&\quad \langle Project_{[ESF]}, Code_{[ESF]}, [0, 1]\rangle, \langle Project_{[ESF]}, Economic\_Field_{[ESF]}, [0, 0.25]\rangle,\\
&\quad \langle Project_{[ESF]}, Type_{[ESF]}, [0, 0.9]\rangle, \langle Project_{[ESF]}, Description_{[ESF]}, [0, 1]\rangle,\\
&\quad \langle Project_{[ESF]}, ESF\_Contribution_{[ESF]}, [0, 0.75]\rangle,\\
&\quad \langle Project_{[ESF]}, Country_{[ESF]}, [0, 1]\rangle, \langle Project_{[ESF]}, Country\_Share_{[ESF]}, [0, 0.9]\rangle\}\\
&nbh(Project_{[ECP]}, 0) = \{\langle Project_{[ECP]}, Country_{[ECP]}, [0, 1]\rangle,\\
&\quad \langle Project_{[ECP]}, Type_{[ECP]}, [0, 0.6]\rangle, \langle Project_{[ECP]}, Dossier\_Number_{[ECP]}, [0, 0.2]\rangle,\\
&\quad \langle Project_{[ECP]}, ESF\_Contribution_{[ECP]}, [0, 0.8]\rangle,\\
&\quad \langle Project_{[ECP]}, Country\_Share_{[ECP]}, [0, 1]\rangle,\\
&\quad \langle Project_{[ECP]}, Public\_Funds_{[ECP]}, [0, 0.9]\rangle,\\
&\quad \langle Project_{[ECP]}, Private\_Funds_{[ECP]}, [0, 1]\rangle,\\
&\quad \langle Project_{[ECP]}, Duration_{[ECP]}, [0, 1]\rangle\}
\end{aligned}$$

μ selects only those arcs of $nbh(Project_{[ESF]}, 0)$ and $nbh(Project_{[ECP]}, 0)$ which participate to the matching and have a similarity coefficient greater than th_μ[4]. Therefore, $\mu(SD, Project_{[ESF]}, Project_{[ECP]}, 0) = [SS_1, SS_2]$, where:

$$\begin{aligned}
&SS_1 = \{\langle Project_{[ESF]}, Country_{[ESF]}, [0, 1]\rangle, \langle Project_{[ESF]}, Type_{[ESF]}, [0, 0.9]\rangle,\\
&\quad \langle Project_{[ESF]}, ESF\_Contribution_{[ESF]}, [0, 0.75]\rangle,\\
&\quad \langle Project_{[ESF]}, Country\_Share_{[ESF]}, [0, 0.9]\rangle\}\\
&SS_2 = \{\langle Project_{[ECP]}, Country_{[ECP]}, [0, 1]\rangle, \langle Project_{[ECP]}, Type_{[ECP]}, [0, 0.6]\rangle,\\
&\quad \langle Project_{[ECP]}, ESF\_Contribution_{[ECP]}, [0, 0.8]\rangle,\\
&\quad \langle Project_{[ECP]}, Country\_Share_{[ECP]}, [0, 1]\rangle\}
\end{aligned}$$

[4] Recall that the similarity coefficient associated to an edge of the bipartite graph is the first coefficient of the corresponding label (see Section 3.2).

The two sets of arcs returned by β_0 could correspond to two unconnected sub-nets; therefore β applies the function ν on them for filtering out unconnected arcs. However, in this case, both SS_1 and SS_2 correspond to rooted, connected sub-nets; therefore, ν does not modify them.

The function β_1 works analogously and returns a further promising pair of sub-nets; the function β_2 does not select any other interesting pair of sub-nets since all target nodes relative to the arcs of $nbh(Project_{[ESF]}, 2)$ and $nbh(Project_{[ECP]}, 2)$ have a similarity degree smaller than th_μ.

In the same way it is possible to compute
$\beta(ESF, ECP, SD, Judicial\_Person_{[ESF]}, Partner_{[ECP]})$ and
$\beta(ESF, ECP, SD, Payment_{[ESF]}, Payment_{[ECP]})$.

3.4 Estimation of the Number of Promising Pairs of Sub-sources

The following theorem provides an estimation of the maximum number of promising pairs of sub-sources which should be examined for a given pair of information sources.

Theorem 1. Let IS_1 and IS_2 be two information sources and let $Net(IS_1)$ and $Net(IS_2)$ be the corresponding SDR-Networks. Let n_{c_1} (resp., n_{c_2}) be the number of complex nodes of $Net(IS_1)$ (resp., $Net(IS_2)$). Let l be the maximum neighborhood index associated to a node of $Net(IS_1)$ or $Net(IS_2)$. Then the number of possible promising pairs of sub-sources is $min(n_{c_1}, n_{c_2}) \times (l + 1)$.
Proof. Immediate, by definition. □

As an example, as for the ESF and ECP information source, considered in the example of Section 3.3, $n_{c_1} = 6$, $n_{c_2} = 5$, $l = 3$; therefore, the number of possible promising pairs of sub-sources is 20.

Actually, since some of the nodes of $Net(IS_1)$ and $Net(IS_2)$ have not a corresponding synonym node and since the maximum neighborhood of some of the nodes is lesser than l, then the actual number of promising pairs of sub-sources relative to ESF and ECP is 9.

4 Derivation of Sub-source Similarities

4.1 Description of the Approach

The proposed technique for constructing the Sub-source Similarity Dictionary SSD consists of two steps. The first one computes the similarity degree relative to each promising pair of sub-sources belonging to the set SPS, returned by the function σ we have seen in the previous section. The second one obtains SSD by selecting only those pairs of sub-sources whose similarity degree is greater than a certain, dynamically computed, threshold. More formally, SSD is constructed as follows:

$$SSD = \sigma_S(\theta(SPS, SD))$$

The function θ receives the set SPS of promising pairs of sub-sources and the Dictionary SD of Synonymies between object classes of involved information sources and returns a set SSS of sub-source similarities. For each pair $[SS_i, SS_j] \in SPS$, θ calls the function ψ' which computes the corresponding similarity degree. More formally,

$$SSS = \theta(SPS, SD) = \{\langle SS_i, SS_j,\ \psi'(SD, \tau'(SS_i), \tau'(SS_j))\rangle \mid [SS_i, SS_j] \in SPS\}$$

The function τ' receives a rooted sub-net SS and returns the nodes of SS:

$$\tau'(SS) = \{N_m \mid \langle N_l, N_m, [d, r]\rangle \in SS\} \cup \zeta(SS)$$

Here, the function ζ receives a rooted sub-net SS and returns its root.

The function ψ' derives the similarity degree associated to SS_i and SS_j by computing a suitable objective function associated to the maximum weight matching on a bipartite graph, constructed from the nodes of SS_i and SS_j.

In more detail, the inputs of ψ' are: *(i)* two sets of nodes $P = \{p_1, \dots, p_r\}$ and $Q = \{q_1, \dots, q_s\}$, *(ii)* a set T of triplets of the form $\langle p_l, q_m, f_{lm}\rangle$, where, for each $p_l \in P$ and $q_m \in Q$, $0 \leq f_{lm} \leq 1$. The output is a value in the real interval $[0, 1]$. Let $BG = (P \cup Q, E)$ be a bipartite weighted graph, where E is the set of weighted edges $\{(p_l, q_m, f_{lm}) \mid f_{lm} > 0\}$. The maximum weight matching for BG, in this case, is a set $E' \subseteq E$ of edges such that, for each node $x \in P \cup Q$, there is at most one edge of E' incident onto x and $\phi'(E') = \sum_{(p_l, q_m, f_{lm}) \in E'} f_{lm}$ is maximum. In this case the objective function associated to the matching is:

$$\psi'(T, P, Q) = \left(1 - \frac{|P|+|Q|-2|E'|}{|P|+|Q|}\right) \frac{\phi'(E')}{|E'|}$$

Here the factor $\frac{1}{|E'|}$ is used to normalize $\phi'(E')$ whereas $(1 - \frac{|P|+|Q|-2|E'|}{|P|+|Q|})$ is exploited for taking into account unmatched nodes. Indeed, $2|E'|$ indicates the number of matched nodes, $|P| + |Q| - 2|E'|$ denotes the number of unmatched nodes, $\frac{|P|+|Q|-2|E'|}{|P|+|Q|}$ indicates the proportion of unmatched nodes. $\psi'(T, P, Q)$ is set to 0 if $E' = \emptyset$. In order to understand the reasoning underlying the definition of ψ', consider the following two situations: *(i)* $|P| = 6$, $|Q| = 8$ and only one edge $(p_l, q_m, 1)$ belongs to E'; *(ii)* $|P| = 1$, $|Q| = 1$ and $E' = \{(p_1, q_1, 1)\}$. In both cases $\phi'(E') = 1$ but, for case *(i)*, $\psi'(T, P, Q) = 0.14$ whereas, for case *(ii)*, $\psi'(T, P, Q) = 1$. This clearly illustrates the need to take unmatched nodes into account.

It is worth pointing out that the maximum weight matching and the corresponding objective function computed here by ψ' are different from those considered by the function ψ described above. Indeed ψ' does not take into account the semantic relevances and, therefore, they cannot influence the computation of the corresponding objective function.

The different behaviour of ψ and ψ' is justified by the following considerations. ψ determines the set of promising pairs of arcs by computing a maximum weight matching on a bipartite graph obtained from the target nodes of the arcs belonging to the neighborhoods of two nodes N_i and N_j. Note that the nodes considered by ψ are *external* to N_i and N_j. Let $N_l \in SDR_1$ be one of these nodes; since we are considering semi-structured information sources, some instances of N_i may not have a corresponding instance of N_l; an analogous reasoning can be drawn for N_j. Therefore the various nodes of SDR_1 (resp., SDR_2) could have different relevances w.r.t. N_i (resp., N_j) (see Section 2) and, consequently, the relevance of each node w.r.t. N_i (resp., N_j) must be taken into account by ψ.

Vice versa, ψ' is called as a step of the computation of the similarity degree between two sub-sources $SS_i \in SDR_1$ and $SS_j \in SDR_2$; the nodes considered by ψ' are *internal* to SS_i and SS_j. Now, all nodes of the sub-source equally contribute to define the overall concept expressed by the sub-source itself. Therefore, no concept of relevance of a node w.r.t. a sub-source can be defined and, consequently, no node relevance must be taken into account in the computation of the similarity degree between sub-sources carried out by ψ'.

After that the set SSS of sub-source similarities has been obtained, the function σ_S is called for constructing the Sub-source Similarity Dictionary SSD by taking those similarities of SSS having a coefficient greater than a certain, dynamically computed, threshold. More formally:

$$SSD = \sigma_S(SSS) = \{\langle SS_i, SS_j, f_{ij}\rangle \mid \langle SS_i, SS_j, f_{ij}\rangle \in SSS, f_{ij} > th_{Sim}\}$$

The threshold th_{Sim} is dynamically computed as:

$$th_{Sim} = min\left(\frac{f_{Sim}^{Max} + f_{Sim}^{Min}}{2}, th_M^{Sim}\right)$$

where f_{Sim}^{Max} (resp., f_{Sim}^{Min}) is the maximum (resp., the minimum) coefficient associated to the similarities of SSS and th_M^{Sim} is a limit threshold value. The formula for th_{Sym} is justified by considering that the technique for selecting the set of promising pairs of sub-sources is defined in such a way that similarities between selected pairs will be probably high. In this context it may happen that *all* pairs of sub-sources are to be considered interesting. We have experimentally set th_M^{Sim} to 0.75.

4.2 A Real Example Case (...Continues)

Consider the SDR-Networks associated to the information sources ESF and ECP shown in Figures 1 and 2. The Sub-source Similarity Dictionary SSD, relative to ESF and ECP, is obtained by computing $\sigma_S(\theta(SPS, SD))$, where SPS is the set of promising pairs of sub-sources constructed by the technique described in Section 3.2 and SD is the Synonymy Dictionary relative to ESF and ECP. In order to obtain the set SSS of sub-source similarities, for each pair $[SS_i, SS_j] \in SPS$, θ calls a function ψ' which computes the corresponding similarity degree.

Due to space constraints, we show the behaviour of θ only for the pair of sub-sources $[SS_1, SS_2] \in SPS$ derived in Section 3.3. In order to compute the similarity degree associated to SS_1 and SS_2, θ calls $\psi'(SD, \tau'(SS_1), \tau'(SS_2))$, where:

$$\begin{aligned}\tau'(SS_1) = \{&Project_{[ESF]}, Country_{[ESF]}, Type_{[ESF]}, ESF\_Contribution_{[ESF]},\\ &Country\_Share_{[ESF]}\}\\ \tau'(SS_2) = \{&Project_{[ECP]}, Country_{[ECP]}, Type_{[ECP]}, ESF\_Contribution_{[ECP]},\\ &Country\_Share_{[ECP]}\}\end{aligned}$$

The value returned by ψ' is $\left(1 - \frac{5+5-2\times5}{10}\right) \times \frac{0.63+1+1+1+1}{5} = 0.93$.
In the same way the similarity degree relative to all the other pairs of SPS is obtained.

SSS is, then, provided in input to the function σ_S which constructs the Sub-source Similarity Dictionary SSD by selecting those similarities of SSS whose coefficient is greater than th_{Sim}. Here $th_{Sim} = min\left(\frac{0.94+0.82}{2}, 0.75\right) = 0.75$; therefore, all similarities of SSS are valid and $SSD = SSS$.

Acknowledgments

The authors gratefully acknowledge the Italian Information System Authority for Public Administration (AIPA) for kindly providing Italian Central Government Office schemes and technical support with them.

References

1. P. A. Bernstein and E. Rahm. Data warehouse scenarios for model management. In *Proc. of International Conference on Entity-Relationship Modeling (ER'00)*, pages 1–15, Salt Lake City, Utah, USA, 2000. Lecture Notes in Computer Science, Springer Verlag. 165
2. P. Fankhauser, M. Kracker, and E. J. Neuhold. Semantic vs. structural resemblance of classes. *ACM SIGMOD RECORD*, 20(4):59–63, 1991. 163
3. Z. Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys*, 18:23–38, 1986. 170
4. W. Gotthard, P. C. Lockemann, and A. Neufeld. System-guided view integration for object-oriented databases. *IEEE Transactions on Knowledge and Data Engineering*, 4(1):1–22, 1992. 163
5. J. A. Larson, S. B. Navathe, and R. Elmastri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, 1989. 163
6. T. Milo and S. Zohar. Using schema matching to simplify heterogenous data translations. In *Proc. of Conference on Very Large Data Bases (VLDB'98)*, pages 122–133, New York City, USA, 1998. Morgan Kaufmann. 163
7. L. Palopoli, D. Rosaci, G. Terracina, and D. Ursino. Un modello concettuale per rappresentare e derivare la semantica associata a sorgenti informative strutturate e semi-strutturate. Atti del Congresso sui Sistemi Evoluti per Basi di Dati (SEBD 2001). In Italian. Forthcoming., 2001. 166
8. L. Palopoli, D. Saccà, G. Terracina, and D. Ursino. A unified graph-based framework for deriving nominal interscheme properties, type conflicts and object cluster similarities. In *Proc. of Fourth IFCIS Conference on Cooperative Information Systems (CoopIS'99)*, pages 34–45, Edinburgh, United Kingdom, 1999. IEEE Computer Society. 163

9. L. Palopoli, G. Terracina, and D. Ursino. A graph-based approach for extracting terminological properties of elements of XML documents. In *Proc. of International Conference on Data Engineering (ICDE 2001)*, pages 330–340, Heildeberg, Germany, 2001. IEEE Computer Society. 165, 166, 167, 171
10. E. Rahm and P. A. Bernstein. On mathing schemas automatically. In *Technical Report MSR-TR-2001-17*, http://www.research.microsoft.com/scripts/pubs/view.asp?TR_ID=MSR-TR-2001-17, 2001. 163, 165
11. N. Rishe, J. Yuan, R. Athauda, S.-C. Chen, X. Lu, X. Ma, A. Vaschillo, A. Shaposhnikov, and D. Vasilevsky. Semantic access: Semantic interface for querying databases. In *Proc. of International Conference on Very Large Data Bases (VLDB 2000)*, pages 591–594, Il Cairo, Egypt, 2000. Morgan Kaufmann. 165
12. D. Rosaci, G. Terracina, and D. Ursino. An algorithm for obtaining a global representation from information sources having different nature and structure. In *Proc. of International Conference on Database and Expert Systems Applications (DEXA 2001)*, Munich, Germany, 2001. Forthcoming. 165
13. S. Spaccapietra and C. Parent. View integration: A step forward in solving structural conflicts. *IEEE Transactions on Knowledge and Data Engineering*, 6(2):258–274, 1994. 163
14. G. Terracina and D. Ursino. Deriving synonymies and homonymies of object classes in semi-structured information sources. In *Proc. of International Conference on Management of Data (COMAD 2000)*, pages 21–32, Pune, India, 2000. McGraw Hill. 164, 165, 166
15. J. A. Wald and P. G. Sorenson. Explaining ambiguity in a formal query language. *ACM Transaction on Database Systems*, 15(2):125–161, 1990. 165

P-Grid: A Self-Organizing Access Structure for P2P Information Systems

Karl Aberer

Distributed Information Systems Laboratory, Department of Communication Systems
Swiss Federal Institute of Technology (EPFL), 1015 Lausanne, Switzerland
karl.aberer@epfl.ch

Abstract. Peer-To-Peer systems are driving a major paradigm shift in the era of genuinely distributed computing. Gnutella is a good example of a Peer-To-Peer success story: a rather simple software enables Internet users to freely exchange files, such as MP3 music files. But it shows up also some of the limitations of current P2P information systems with respect to their ability to manage data efficiently. In this paper we introduce P-Grid, a scalable access structure that is specifically designed for Peer-To-Peer information systems. P-Grids are constructed and maintained by using randomized algorithms strictly based on local interactions, provide reliable data access even with unreliable peers, and scale gracefully both in storage and communication cost.

Keywords. Peer-To-Peer computing, Distributed Indexing, Distributed Databases, Randomized Algorithms.

1 Introduction

Peer-To-Peer (P2P) systems are driving a major paradigm shift in the era of *genuinely* distributed computing [2]. Major industrial players believe "P2P reflects society better than other types of computer architectures. It is similar to when in the 1980's the PC gave us a better reflection of the user" (www.infoworld.com).

In a P2P infrastructure, the traditional distinction between clients and backend (or middle tier application) servers is simply disappearing. Every node of the system plays the role of a client and a server. The node *pays* its participation in the global exchange community by providing access to its computing resources. Gnutella is a good example of a P2P success story: a rather simple software enables Internet users to freely exchange files, such as MP3 music files.

In the current P2P file-sharing systems, like Gnutella [1], no indexing mechanisms are supported. Search requests are broadcasted over the network and each node receiving a search request scans its local database (i.e. its file system) for possible answers. This approach is extremely costly in terms of communication and leads to high search costs and response times. For supporting efficient search, however, appropriate access structures are prerequisite.

Access structures in distributed information systems have been addressed in the area of distributed and parallel databases. Different approaches to data

© Springer-Verlag Berlin Heidelberg 2001

access in distributed environments have been taken. We mention some of the principal approaches.

- The distribution of one copy of a search tree over multiple, distributed nodes is a technique that has been investigated in [6]. The same authors have shown that, under certain assumptions, with that approach balanced search trees do not exist [7].
- The replication of the complete search structure is an approach that underlies the RP^*-Trees proposed in [8]. In [10] a mechanisms is proposed that leads eventually to the replication of the search structures.
- Scalable replication of a search tree (more precisely B-Tree) is proposed in [5] (dB-Tree) and [11] (Fat-BTree). With scalable replication each node stores a single leaf node of the search tree, the root node of the search tree is replicated to every node, and the intermediate nodes are replicated such that each node maintains a path from the root to its own leaf.
- No search structures: in these approaches operation messages are broadcasted to all participating nodes. E.g. with RP^*_N [8] the data is range partitioned as in B-Trees but no index exists and a multicast mechanism is used. In current P2P file sharing systems, like Gnutella, the P2P network is used to propogate search requests to all reachable peers.

Many of these approaches assume a reliable execution environment, require some centralized services, are designed for a fairly small numbers of nodes (hundreds) and focus on deterministic execution guarantees. In this paper we would like to take a different approach and address the question of how an access structure can be built by a community of a very large number of unreliable peers without any central authority, that can provide still a certain level of reliability of search and scales well in the number of peers, both in storage and communication cost. In order to obtain scalability we use the approach of scalable replication of tree structures, as proposed in [5][11]. To construct these search structures and perform searches and updates we use randomized algorithms, that are based exclusively on local interactions among peers. The idea is that by randomly meeting among each other the peers successively partition the search space and retain enough information in order to be able to contact other peers for efficiently answering future search requests. The resulting distributed access structure we call *P-Grid* (Peer-Grid). As this paper addresses the prinicipal feasibility of such an approach we make the simplifying assumption that the data distribution is not skewed. Thus the use of binary search trees over a totally order domain of keys is sufficient (as e.g. also used in [6]). Similarly, for the analysis and simulation of P-Grids, we make uniformity assumptions on the peer behavior. Though definitely in a next step skewed data distributions and extended methods for balancing the search trees, as well known from B-Tree structures, are required, even the basic methods in this paper can be beneficial to improve the efficiency of current file sharing applications. The main characteristics of P-Grids are that

- they are completely decentralized, there exists no central infrastructure, all peers can serve as entry point to the network and all interactions are strictly local.

– they use randomized algorithms for constructing the access structure, updating the data and performing search; probabilistic estimates can be given for the success of search requests, and search is robust against failures of nodes.
– they scale gracefully in the total number of nodes and the total number of data items present in the network, equally for all nodes, both, with respect to storage and communication cost.

In the next section we introduce our system model and the structure of P-Grids. In Section 3 we describe the distributed, randomized algorithm that is used to construct P-Grids. In Section 4 we give some analysis on basic properties of P-Grids. Section 5 contains extensive simulation results, that demonstrate the feasibility of the P-Grid algorithms and exhibit important scalability properties of P-Grids. Section 6 indicates directions for future work.

2 System Model and Access Structure

We assume that a community of peers P is given that can be addressed using a unique address $addr : P \rightarrow ADDR$. For an address $r \in ADDR$ we define $peer(r) = a$ iff $addr(a) = r$ for $a \in P$. The peers are online with a probability $online : P \rightarrow [0, 1]$. Peers that are online can be reached reliably through their addressusing the underlying communication infrastructure.

Every peer stores information items from a set DI that are characterized by an index term from a set K. The set of index terms is totally ordered, such that a search tree can be constructed in the usual way. In the following, we assume that the index terms are binary strings, built from 0's and 1's and that a key $k = p_1 \dots p_n,\ p_i \in \{0, 1\}$ corresponds to a value $val(k) = \sum_{i=1}^{n} 2^{-ip_i}$ and an interval $I(k) = [val(k), val(k) + 2^{-n}[\ \subseteq\ [0, 1[$.

Now we define the access structure. The goal is to construct an access structure, such that

– The search space is partitioned into intervals of the form $I(k),\ k \in K$. Every peer takes over responsibitity for one interval $I(k)$. As each key corresponds to a path in the binary search tree we will also say that the peer is responsible for the *path* k.
– Taking over responsibility for an interval $I(k)$ means, that a peer should provide the addresses of all peers that have an information item with a key value k_{query} that belongs to $I(k)$, i.e. $val(k_{query}) \in I(k)$.
– For each prefix k_l of k of length $l,\ l = 1, \dots, length(k)$ a peer a maintains references to other peers, that have the same prefix of length l, but a different value at position $l+1$, for the key they are responsible for. We will call these references to other peers, a's references at level $l + 1$. These references are used to route search requests for keys with prefix k_l, but a continuation that does not match the own key, to other peers.
– A search can start at each peer.

Before giving the formal definition of the access structure, we give in Fig. 1 a simple example. The different levels relate to the different levels of the binary search tree. The intervals relate to the nodes of the search tree. We indicate the key values that correspond to the intervals (i.e. 0 and 1 at level 1, 00, 01, 10, and 11 at level 2). At the lowest level we entered have 6 peers, indicated by black circles, into the leaf nodes corresponding to the keys they are responsible for. Multiple peers can be responsible for the same key. We will call these later also *replicas*. We also entered the agents into all intervals on the path from the root to their leaf node. At the leaf nodes we show the pointers to specific data items whose key is a prefix of the key of the peer.

At level zero every peer is associated with the whole interval, in other words, it stores a root node of the search tree. At level 1 every peer is associated with exactly one of the two intervals. At level 2 every peer is associated with exactly one interval. The connectors from one level to the next are the references that s peer maintains to cover the other side of the search tree. For example, at level 0 peer 1 has a reference to peer 3, and at level 1 peer 1 has a reference to peer 2.

When a search request is issued it is routed over the responsible peers. There are two possiblities, either at the next level the peer itself is responsible, then it can further process the request itself, or, the request needs to be forwarded to another peer. For illustration we have included into Fig. 1 the processing of two queries. In the first example the query 00 is submitted to peer 1. As peer 1 is reponsible for 00 it can process the complete query. In the second example the query 10 is submitted to peer 6. Using its reference at level 0 it contacts peer 5, which in turn contacts at level 1 peer 4, who is responsible for the key corresponding to the query.

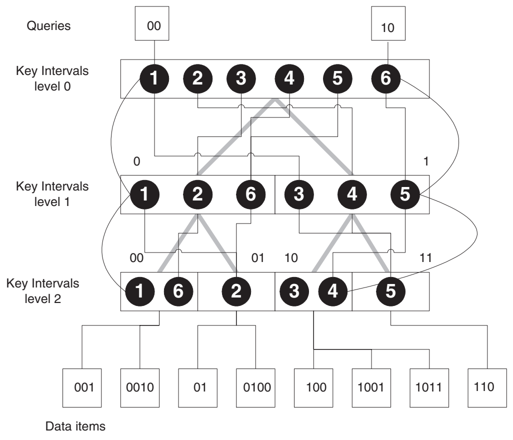


Fig. 1. Example P-Grid

We define now formally the data structure for peers that allows to represent the P-Grid. Every peer $a \in P$ maintains a sequence

$$(p_1, R_1)(p_2, R_2) \ldots (p_n, R_n),$$

where $p_i \in \{0,1\}$ and $R_i \subset ADDR$. We define $path(a) = p_1 \ldots p_n$, $prefix(i,a) = p_1 \ldots p_i$ for $0 \leq i \leq n$ and $refs(i,a) = R_i$. In addition the number of references in R_i will be limited by a value $refmax$. The sets R_i, $1 \leq i \leq n$ are references to other peers and satisfy the following property:

$$r \in refs(i,a) : prefix(i, peer(r)) = prefix(i-1,a)p_i^-$$

where $path(a) = p_1 \ldots p_n$ and p^- is defined as $p^- = (p+1)\ MOD\ 2$. In addition, each peer maintains a set of references $D \subset ADDR \times K$ to the peers that store data items indexed by keywords k for which $path(a)$ is a prefix. In other words, at the leaf level the peer knows at which peers data items corresponding to the search keys that it is responsible for, can be found.

This gives rise to a straightforward depth first search algorithm shown in Fig. 2. Given a P-Grid, a query p can be issued to each peer a through a call $query(a,p,0)$.

```
query(a, p, l)
{
found = FALSE;
rempath = sub_path(path(a), l+1, length(path(a)));
compath = common_prefix_of(p, rempath);
IF length(compath) = length(p)
   OR length(compath) = length(rempath) THEN
  result = a; found = TRUE ELSE
  IF length(path(a)) > l + length(compath) THEN
     querypath = sub_path(p, length(compath) + 1, length(p));
     refs = refs(l + length(compath) + 1, a);
     WHILE |refs| > 0 AND NOT found
        r = random_select(refs);
        IF online(peer(r))
           found = query(peer(r), querypath, l + length(compath));
RETURN found;

/* Comment:
sub_path(p1...pn, l, k) := pl...pk
common_prefix_of(p1...pk pk+1...pn, p1...pk qk+1...ql = p1...pk)
random_select(refs) returns a random element from refs
and removes it from refs */
}
```

Fig. 2. P-Grid search algorithm

3 P-Grid Construction

Having introduced the access structure and the search algorithm, the main question is now, of how a P-Grid can be constructed. As there exists no global control this has to be done by using exclusively local interactions. They idea is that whenever peers meet, they use the opportunity to create a refinement of the access structure. We do not care at this point why peers meet. They may meet randomly, because they are involved in other operations, or because they systematically want to build the access structure. But assuming that by some mechanisms they meet frequently, the procedure works as follows.

Initially, all peers are responsible for the whole search space, i.e. all search keys. At that stage, when two peers meet initially, they decide to split the search space into two parts and take over responsibility for one half each. They also store the reference to the other peer in order to cover the other part of the search space. The same happens whenever two peers meet, that are responsible for the same key at the same level.

However, as soon as the P-Grid develops, also other cases occur. Namely, peers will meet where their keys share a common prefix or where their keys are in a prefix relationship. In the first case, what the peers can do, is to initiate new exchanges, by forwarding each other to peers they are themselves referencing. In the second case the peer with the shorter key can specialize by extending its key. In order to obtain a balanced P-Grid it will specialize in the opposite way the other peer has already done at that level. The other peer remains unchanged. These considerations give rise to the algorithm in Fig. 3 that two peers $a1$ and $a2$ execute when they meet.

A few remarks are in place on this algorithm. A measure to prevent overspecialization of peers is to bound the maximal length of paths that can be constructed to *maxl*. Simulations show that this results in a more uniform distribution of path lengths among peers and better convergence of the P-Grid. Also such a bound is needed when a certain degree of replication at the lowest grid level is to be achieved. The disadvantage is that some global knowledge is used, namely the maximal path length, which not always might be locally available or easily derivable. In practical applications, one possible indication that a path has reached *maxl* could be that the number of data items belonging to the key is falling below a certain threshold.

An alternative would be to avoid overspecialization by taking another approach in Case 2 and Case 3, where one path is subpath of the other and the peer with shorter path chooses to specialize differently than the other peer. Here one could shorten the longer path if the difference in length is greater than 1, such that both resulting paths have the same length. However, this would require additional means to maintain consistency of references as peers give up responsibility for keys by generalizing and could possibly specialize at a later stage differently. This approach also slows down convergence. So we ommitted this possibility here.

The recursive executions of the exchange function by using the locally available references (Case 4) have an important influence on the performance of the

```
exchange(a1, a2, r)
{
    commonpath = common_prefix_of(path(a1), path(a2));
    lc = length(commonpath);
    IF lc > 0
      (* exchange references at level where the paths agree *)
      commonrefs = union(refs(lc, a1), refs(lc, a2));
      refs(lc, a1) = random_select(refmax, commonrefs);
      refs(lc, a2) = random_select(refmax, commonrefs);
      l1 = length(sub_path(path(a1), lc + 1, length(path(a1)));
      l2 = length(sub_path(path(a2), lc + 1, length(path(a2)));
      (* Case 1: both paths empty, introduce new level *)
      CASE l1 = 0 AND l2 = 0 AND length(commonpath) < maxl
         path(a1) = append(path(a1), 0);
         path(a2) = append(path(a2), 1);
         refs(lc + 1, a1) = {a2};
         refs(lc + 1, a2) = {a1};
      (* Case 2: one remaining path empty, split shorter path *)
      CASE l1 = 0 AND l2 > 0 AND length(commonpath) < maxl
         path(a1) = append(path(a1), value(lc+1, path(a2))^-;
         refs(lc + 1, a1) = {a2};
         refs(lc + 1, a2) =
               random_select(refmax, union({a1}, refs(lc+1, a2));
      (* Case 3: analogous to case 2 *)
      CASE l1 > 0 AND l2 = 0 AND length(commonpath) < maxl
         path(a2) = append(path(a2), value(lc+1, path(a1))^-;
         refs(lc + 1, a2) = {a1};
         refs(lc + 1, a1) =
               random_select(refmax, union({a2}, refs(lc+1, a1));
      (* Case 4: recursively exchange with referenced peers *)
      CASE l1 > 0 AND l2 > 0 AND r < recmax,
         refs1 = refs(lc+1, a1)~{a2};
         refs2 = refs(lc+1, a2)~{a1};
         FOR r1 IN refs1 DO
            IF online(peer(r1)) THEN exchange(a2, peer(r1), r+1);
         FOR r2 IN refs2 DO
            IF online(peer(r2)) THEN exchange(a1, peer(r2), r+1);

/* Comment: random_select(k, refs) returns a set with k random
            elements from refs.
            append(p1...pn, p) = p1...pn p
            value(k, p1...pn) = pk
            p^- = 1+p MOD 2 */
}
```

Fig. 3. P-Grid construction algorithm

method. These executions are more promising of ending up in a successful specialization as they are already targetted to a more specific set of candidates. On the other hand the recursive executions might lead to a quick overspecialization of the P-Grid for subregions of the search tree. Therefore, we bound the recursion depth up to which the exchange function is called by the value *recmax*. This value has a very strong influence on the global performance of the algorithm, as we will see later.

So far, we have only considered the construction process of the access structure itself. At the leaf level the peers need also to know the data items, respectively the peers storing those data items, that correspond to their responsibility. As many peers can be responsible for the same key the general problem is of how to find all those peers in case of an update. Different strategies are possible:

- Randomly performing depth first searches for peers responsible for the key multiple times and propagating the update to them
- Performing breadth first searches for peers responsible for the key once and propagating the update to them
- Creating a list of buddies for each peer, i.e. other peers that share the same key, and propagate the update to all buddies.

We will not give the detailed algorithms here as they are quite obvious. But in Section 5 we will identify by using simulations, which is the most efficient method.

4 Analysis of Search Performance

We want to analyze the question of how probable it is to find a peer that is responsible for a specific search key starting the search at an arbitrary peer. This analysis allows to give rough estimates on the sizing of the P-Grid parameters that are required to achieve a desired search reliability in a given setting. We perform the analysis for an idealized situation, where for all peers the parameters of the P-Grid, like keylength and number of peers responsible for the same key, are uniformly distributed. Though such a distribution will not occur in practice it gives a good estimation the quantitative nature of a P-Grid.

The following parameters determine the problem. The number of peers N and the number of data objects each peer can store d_{peer} determine the total number of data objects that can be stored in the network as $d_{global} = N * d_{peer}$. The size of a reference r and the amount of space each peer is willing to make available for indexing purposes s_{peer} determines the possible number of references that can be stored at each peer $i_{peer} = \frac{s_{peer}}{r}$.

Now we determine the number of entries required for a certain grid organisation. The length of a key required to differentiate data items located at different peers is given by

$$k \geq log_2 \frac{d_{global}}{i_{leaf}} \tag{1}$$

where i_{leaf} is the number of references to data items each peer stores at the leaf level.

Then the total number of index entries stored at a peer is given by $i_{leaf} + k * refmax$, where $refmax$ is the multiplicity of references used to build the grid structure. Thus we obtain the constraint $i_{leaf} + k * refmax \leq i_{peer}$ which determines the value of i_{leaf}. In order to allow at the lowest level of the grid the support for $refmax$ alternative peers, references to data items need to be replicated with a factor of $refmax$. This is only possible if

$$\frac{d_{global}}{i_{leaf}} * refmax \leq N \tag{2}$$

i.e. there must be sufficiently high numbers of peers available, such that each interval at the lowest grid level is supported by at least $refmax$ peers.

Given a constant probability p that a peer is online we are now interested in the question what is the probability of performing a successful search for a peer that is responsible for the query key. In the worst case at each level of the grid a new peer needs to be contacted, that is selected out of the available references. At each level of the grid, the probability of reaching a peer at the next level is $1-(1-p)^{refmax}$, i.e. one minus the probability that all $refmax$ referenced peers are offline. Since the search tree is of depth k, then the probability of performing a successful search for a key is

$$(1-(1-p)^{refmax})^k \tag{3}$$

We give now an example, to illustrate what a P-Grid would cost in terms of space for a practical setting.

Example. Let us consider the setting P2P file sharing as it currently is found with Gnutella. We use some rough estimates of the actual parameters that are observed for this application. Assume that $d_{global} = 10^7$ data objects (files) exist, that a reference costs at most $r = 10$ Bytes of storage (the path plus the IP address) and that every peer is willing to provide $s_{peer} = 10^5$ Bytes of storage for indexing (which is in fact far less than the size of an average media file). Furthermore, we assume that peers are online with probability 30%.

Let us now analyze how large a community for supporting the 10^7 files must be in order to ensure that search requests for files are answered with a probability of 99%. Each peer can store at most $i_{peer} = 10^4$ references. If we "guess" a value of $i_{leaf} = 10^4 - 200$, we see that inequality (1) is satisfied for a value of $k = 10$. For a value of $refmax = 20$ then, according to (3), the probability of sucessfully finding a peer for a given key is larger than 99%. The storage required is due to our good initial guess exactly $s_{peer} = 10^5$. The number of peers required to support this grid has to be according to inequality (2) larger than 20409. This is a very reasonable number compared to the size of the actual Gnutella user community.

5 Simulation

For performing simulations we have implemented the algorithm for constructing P-Grids using the mathematical computing package Mathematica. We intend to obtain results on the following questions.

- How many communications in terms of executing the exchange algorithm are required for building a P-Grid ?
- What is the influence of the recursion factor *recmax* in the exchange algorithm on the efficiency of the P-Grid construction ?
- Is the resulting structure reasonably well balanced with respect to distribution of path lengths and number of replicas per path ?
- How reliable can data be found using the P-Grid ?

5.1 Performance of the Construction Method

In the following simulations we analyze the convergence speed when the P-Grid is constructed. The peers meet randomly pairwise and execute the exchange function. We consider a P-Grid as constructed when the average length of the keys that the peers are responsible for reaches a certain threshhold t, i.e. $\frac{1}{N}\sum_{a \in P} length(path(a)) < t$. In the following the path length is bounded by $maxl = 6$ and the threshold is 99% of $maxl$ (5.94). We count the number of calls to the exchange function (e) to determine the construction cost.

First we investigate the relationship between the number of peers and construction cost. We vary the number of peers from 200 to 1000. We use a recursion depth $recmax$ of 0 and 2. The value of $refmax$ was set to 1, i.e. only one reference to another peer ist stored. This influences the performance in the case where $recmax > 0$. The results indicate that a linear relationship exists between the number of peers (N) and the total number of communications (e) needed in building the P-Grid. As a consequence, every peer has to perform on average a (practically) constant number of exchanges to reach its maximal path length independently of the total number of peers involved.

	recmax = 0		recmax=2	
N	e	$\frac{e}{N}$	e	$\frac{e}{N}$
200	15942	79.71	4937	24.68
400	27632	69.08	10383	25.95
600	43435	72.39	15228	25.38
800	59212	74.01	18580	23.22
1000	74619	74.61	25162	25.16

The next simulation shows how the choice of a maximal path length $maxl$ influences the number of exchanges e_{maxl} required. The simulation is done for $N = 500$ peers. The results indicate that the number of communications grows exponentially in the path length when no recursion is used. With a recursion bound $recmax = 2$ the convergence speeds up substantially.

	$recmax = 0$			$recmax = 2$		
$maxl$	e_{maxl}	$\frac{e_{maxl}}{N}$	$\frac{e_{maxl}}{e_{maxl-1}}$	e_{maxl}	$\frac{e_{maxl}}{N}$	$\frac{e_{maxl}}{e_{maxl-1}}$
2	4893	9.78		5590	11.18	
3	9780	19.56	1.998	7289	14.57	1.303
4	18071	36.14	1.847	8215	16.43	1.127
5	35526	71.05	1.965	13298	26.59	1.618
6	72657	145.31	2.045	17797	35.59	1.338
7	171770	343.54	2.364	27998	55.99	1.573

The following simulation shows that the recursion depth *recmax* has substantial influence on the convergence speed. When using recursive calls to the exchange function the probability that a random meeting leads to a successful exchange increases. However, if recursion is not constrained this can lead to negative effects as the peers tend to overspecialize quickly. The result shows that for 500 peers and maximal path length 6 the optimal recursion depth limit is 2.

$recmax$	e	$\frac{e}{N}$
0	35436	70.87
1	15377	30.75
2	12735	25.47
3	16595	33.19
4	18956	37.91
5	22426	44.85
6	25130	50.26

If $refmax > 1$, i.e. peers maintain more than one reference to other peers at each level, i.e. there exist more possibilities to make recursive calls. Thus if $recmax > 0$ this should have an influence on the number of exchanges that are performed when constructing the P-Grid. Note that this additional effort is rewarded by a higher density of the P-Grid. We analyzed this with a simulation with $N = 1000$ peers and a recursion depth limit $recmax = 2$.

$refmax$	e	$\frac{e}{N}$
1	25285	25.28
2	39209	39.20
3	72130	72.13
4	125727	125.72

As one can see the number of exchanges grows exponentially, which is undesirable. In fact, this turned out to be a weakness in the algorithm we proposed. However, there exists a simple way to fix this. One limits the number of referenced peers with which exchanges are made throughout recursion. Then the results become very stable as the following simulation shows, where recursive calls are only made to 2 randomly selected referenced peers.

$refmax$	e	$\frac{e}{N}$
1	23826	23.826
2	37689	37.689
3	40961	40.961
4	43914	43.914

5.2 Search and Update Performance

The subsequent simulations are based on a configuration that is similar to the one described in the example in Sec. 4. This confirms that the algorithms scale well for realistic parameter settings. We use 20000 peers that build a P-Grid with keys of maximal length 10. The maximal number of references $refmax$ at each level is limited to 20. The online probability of peers is 30%.

Building a P-Grid of that size within a simulation environment requires considerable computing resources. Within approximately 10 hours of running time on a Pentium III processor the P-Grid was constructed up to an average depth of 9.43. During that time 1250743 exchanges among peers were performed, i.e. 62 per peer. Based on this P-Grid we make the following observations.

First we see that the replicas, i.e. the number of different peers responsible for the same key are fairly uniformly distributed. Figure 4 shows this result. The x-axis indicates the replication factor and the y-axis the number of peers that have this replication factor. The average number of replicas for a peer is 19.46.

A simple, intuitive argument shows that this is not surprising as the exchange function inherently tends to balance the distribution of keys. For example, a peer will decide upon the first bit of the key when it meets the first time another peer that has already taken this decision or also needs to decide on the first bit. It will decide in both cases opposite to the other peer. Thus, if there exists an imbalance in the distribution of the first bit, this is compensated for.

Next we would like to confirm the analysis of Sec. 4. We search 10000 times for a random key of length 9. Only 30% of the peers are online. In 99.97% of the cases the search was successful and a search required on average 5.5576 messages among peers. We were counting as messages the successful calls of the query operation to another peer. This shows that searches can be performed reliably.

Now we turn to the question of how updates can be performed. The problem with an update, in contrast to a search, is, that we have to find all replicas of a path, not just one. Therefore we analyze how efficiently a large fraction of all replicas can be found. We compare the three approaches of (1) repeated depth

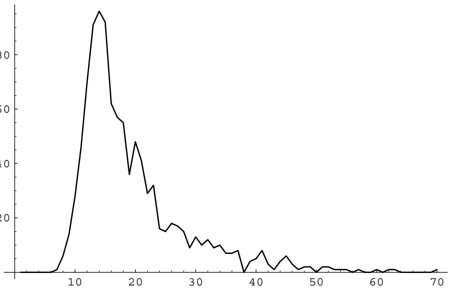


Fig. 4. Replica distribution

first searches, (2) repeated depth first searches including all buddies that have been identified throughout index construction, and (3) repeated breadth first searches.

We repeatedly searched for a random key of length 9 and then computed the fraction of replicas identified as compared to the actual number of existing replicas. Figure 5 shows the result. The x-axis shows the number of messages used in the insertion process, and the y-axis the percentage of successfully identified replicas. One can see that clearly the strategy of using breadth first searches is by far superior, while the two other methods perform comparably.

One can see also, that for achieving a high update reliability a fairly large number of messages is required (hundreds per updated replica). So this approach is acceptable where updates are rare as compared to queries. However, the relevant problem is not to achieve high reliability in keeping the replicas consistent, but rather high reliability in obtaining correct query results. Thus we can use a different approach. We update a sufficiently high number of replicas using fewer messages, and use repeated query operations. Obviously, if more than half of the replicas are correct, by repeating queries, arbitrarily high reliability can be achieved by a making majority decision. In this manner we increase slightly the query cost and in turn reduce drastically the update cost. There is another factor that is helpful in that context. Not all replicas are as likely to be found. This implies that replicas that are found during updates are also more likely to be found during queries.

We perform a simulation to illustrate the tradeoff among update and query cost and indicate the optimal combinations of update and query strategies. 100 updates were performed and each updated data item was searched 10 times, thus 1000 queries were performed in each configuration. Updates are performed using breadth first search, where at each level *recbreadth* references are followed. The search was repeated *repetition* times. The *successrate* is the fraction of successfully answered queries after an update. The cost is in terms of number of messages.

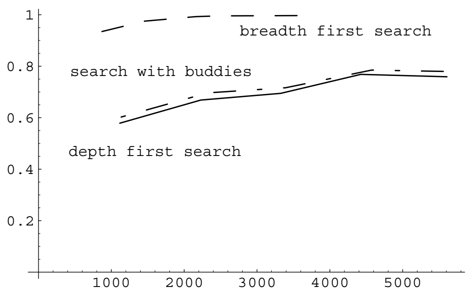


Fig. 5. Finding all replicas

The simulation shows that the approach of using repeated searches to achieve query reliability pays off dramatically. The configuration *recbreadth* = 3 and *repetition* = 3 without using repeated search, which achieves only 99,4% reliability, would require at least a ratio of 160 queries per update in order reach the break-even point compared to the configuration *recbreadth* = 2 and *repetition* = 3 using repeated search, which offers practically 100% reliability.

repetitive search				
recbreadth	repetition	successrate	query cost	insertion cost
2	1	1	137	78
2	2	1	34	147
2	3	1	17	224
3	1	1	112	637
3	2	1	13	1434
3	3	1	13	2086
non-repetitive search				
2	1	0.65	5.5	72
2	2	0.85	5.6	145
2	3	0.89	5.4	212
3	1	0.95	5.5	734
3	2	0.98	5.5	1363
3	3	0.994	5.4	2080

6 Discussion

This paper introduced P-Grid, a first step towards developing scalable, robust and self-organizing access structures for P2P information systems. To illustrate the effectiveness of a P-Grid we can compare it to centralized replicated server architectures. Assume D is the number of data items and N the number of peers. For storage we consider the number of references to be stored at the nodes ignoring local indexing cost. For querying we consider the number of messages exchanged assuming that each node creates a constant number of queries per time unit. Then a solution using centralized replicated servers compares to P-Grid as follows.

	P-Grid	Central Server
Storage	peers: $O(\log D)$	server: $O(D)$
		client: constant
Query	peers: $O(\log N)$	server: $O(N)$
		client: constant

One can see that both storage and communication cost scale well for the P-Grid. For a centralized servers the linear growth of communication cost in the client number is critical as servers become bottlenecks. This shows that besides robustness also scalability is an asset of P-Grids.

The approach presented in this paper is limited to uniform data distributions. For uniformly distributed key values the P-Grid can be immediately applied. For prefix search on text the algorithm can be adapted by extending the $\{0, 1\}$

alphabet. This would allow to directly support trie search structures. However the decentralized support for more sophisticated search structures, like in [3] is a challenging research topic.

An obvious continuation of this research is to develop P-Grids that can support skewed data distributions. To that extent throughout the construction process the actual data distribution needs to be taken into account and the structures have to continuously adapt to updates. Another natural extension of the approach would be to take system parameters, like known reliability of peers, knowledge on the network topology or knowledge on query distribution into account for optimizing P-Grid construction and updates. To achieve load balancing a computational economy can be imposed, as already investigated for distributed data mangement in [4][9].

We see the P-Grid, as it is presented in this paper, as a first representative of access structures for P2P information systems, for which many variations will be developed, that are adapted to the specific requirements of various P2P application domains.

Acknowledgements

I would like to thank Manfred Hauswirth and Roman Schmidt for carefully reading and commenting the manuscript. I would also like to thank Magdalena Punceva and Rachid Guerraoui for many helpful discussions. This work also greatly benefited from the inspiring working environment that is provided by the newly founded Communication Systems Department at EPFL.

References

1. E. Adar and B. A. Huberman: *Free riding on Gnutella* Technical report, Xerox PARC, 10 Aug. 2000. 179
2. D. Clark. *Face-to-Face with Peer-to-Peer Networking.* IEEE Computer, January 2001. 179
3. Y. Chen, K. Aberer: *Combining Pat-Trees and Signature Files for Query Evaluation in Document Databases* DEXA 99, LNCS, Vol. 1677, p. 473-484, Springer, 1999. 193
4. D. F. Ferguson, C. Nikolaou and Y. Yemini *An Economy for Managing Replicated Data in Autonomous Decentralized Systems* International Symposium on Autonomous Decentralized Systems (ISADS'93), 1993. 193
5. T. Johnson, P. Krishna *Lazy Updates for Distributed Search Structure* ACM SIGMOD 93, p. 337-346, 1993. 180
6. B. Kröll, P. Widmayer *Distributing a Search Tree Among a Growing Number of Processors.* ACM SIGMOD 94, p. 265-276, 1994. 180
7. B. Kröll, P. Widmayer *Balanced Distributed Search Trees Do Not Exist* WADS 95, p 50-61, 1995. 180
8. W. Litwin, M. Neimat, D. A. Schneider *RP*: A Family of Order Preserving Scalable Distributed Data Structures.* VLDB 94, p. 342-353, 1994. 180

9. M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, Jeff Sidell, Carl Staelin, Andrew Yu: Mariposa *A Wide-Area Distributed Database System* VLDB Journal 5(1): 48-63, 1996. 193
10. R. Vingralek, Y. Breitbart, G. Weikum *SNOWBALL: Scalable Storage on Networks of Workstations with Balanced Load* Distributed and Parallel Databases Vol 6(2), Kluwer Academic Publishers, 1998. 180
11. H. Yokota, Y. Kanemasa, J. Miyazaki *Fat-Btree: An Update-Conscious Parallel Directory Structure* ICDE 99, p. 448-457, 1999. 180

Moving Active Functionality from Centralized to Open Distributed Heterogeneous Environments

Mariano Cilia [*], Christof Bornhövd [**], and Alejandro P. Buchmann

Databases and Distributed Systems Group, Department of Computer Science
Darmstadt University of Technology - Darmstadt, Germany
{cilia,bornhoev,buchmann}@informatik.tu-darmstadt.de

Abstract. Active functionality is especially useful for enforcing business rules in applications, such as Enterprise Application Integration (EAI) and e-commerce. It can be used as glue among existing applications, and for data transformations between heterogeneous applications. However, traditional active mechanisms have been designed for centralized systems and are monolithic, thus making it difficult to extend and adapt them to the requirements imposed by distributed, heterogeneous environments. To correct this we present a flexible, extensible, service-based architecture built on ontologies, services and events/notifications. The main contributions of this work are: i) the homogeneous use of ontologies for a semantically meaningful exchange and combination of events in open heterogeneous environments, and for the infrastructure itself; ii) a flexible architecture for the composition of autonomous, elementary services to provide Event-Condition-Action (ECA) functionality in different configurations; iii) the interaction of these services via notifications using a publish/subscribe mechanism (concept-based addressing).

1 Introduction

Active database systems (aDBMS) enhance traditional database functionality with powerful rule processing capabilities. The database system performs certain operations automatically in response to events occurring and certain conditions being satisfied. Active functionality is especially useful for enforcing business rules. However, traditional active mechanisms have been designed for centralized systems and are monolithic, thus making it difficult to extend or adapt them. New large-scale applications, such as EAI, e-commerce or Intranet applications impose new requirements. In these applications, integration of different subsystems and collaboration with partners' applications is of particular interest, since business rules are out of the scope of a single application. For example, consider the following business rule: "when the volume of corn falls below 1200 tons, and the corn comes from Euroland, and Euro/US$ is under 0.8 then bid for all at 2.30 US$ per ton, and also notify the manager". Involved events and

[*] Also ISISTAN, Faculty of Sciences, UNICEN, Tandil, Argentina
[**] Current affiliation: Hewlett-Packard, e-Solutions Divison, CA, Cupertino

© Springer-Verlag Berlin Heidelberg 2001

data are coming from diverse sources, here and also the execution of actions is performed on different (sub-) systems. Observe that events and actions are not necessarily directly related with database operations.

In such scenarios, the required active functionality should be moved outside of the active database system by offering a flexible service that runs decoupled from the database, and that can be combined in many different ways and used in a variety of environments. A service-based architecture seems to be appropriate, in which an active functionality (ECA rule) service can be seen as a composition of other services, like event detection, event composition, condition evaluation, and action execution. Thus, services can be combined and configured according to the required functionality, as proposed by the unbundling approach in the context of aDBMSs [11,12,18]. Whether the unbundling approach is realistic for database systems or not, it is inadequate for distributed environments since DBMS components to be "unbundled" are designed with homogeneous, centralized environment in mind. Combining components developed by different, independent providers leads inevitably to problems if the meaning of terms employed by different components and data to be exchanged is not shared. A similar problem is encountered when integrating heterogeneous applications.

In addition to the difficulties introduced by heterogeneity, the inherent characteristics imposed by large-scale distributed environments, in particular, impact the following issues of active functionality: event exchange mechanism, event semantics (correct interpretation and use of events), event filtering, and complex event detection.

In this paper we present a service-based architecture that supports the use of active functionality in various environments. In this context we focus on aspects related with the problems of interpreting event contents coming from different systems and sources. Despite the fact that event composition is an important issue we do not present here details about complex event detection since they are out of the scope of this paper.

The rest of this paper is organized as follows. Section 2 provides a discussion of the most important aspects and proposals for moving from centralized active functionality to distributed heterogeneous environments. In Section 3 the conceptual foundation of our proposal for a flexible and extensible active service for this kind of environment is presented. In Section 4 our reference architecture is introduced. Finally, in Section 5 we present conclusions, address open issues, and discuss future work. Examples throughout the paper are related to online auctions.

2 Moving to Open Distributed Heterogeneous Environments

Active database functionality is developed for a particular DBMS, becoming part of a large monolithic piece of software (the DBMS itself). Monolithic software is difficult to extend and adapt. Moreover, active functionality tightly coupled to a concrete database system hinders its adaptation to today's Internet applications, such as, e-commerce, where heterogeneity and distribution play a significant role but are not directly supported by traditional (active) databases [18].

Another weakness of tightly coupled aDBMSs is that active functionality can not be used on its own, without the full data management functionality. However, active functionality is also needed in applications that require no database functionality at all, or only simple persistence support. As a consequence, active functionality should be offered not only as part of the DBMS, but also as a separate service which can be combined with other services to support Internet-scale applications. Implications of heterogeneity, distribution and active functionality as an independent service, and a review of other approaches that address these issues are discussed in the following subsections.

2.1 Heterogeneity

The integration of events coming from heterogeneous sources is comparable to the problems faced in federated multi-database systems. Similar to heterogeneity in data, we also have heterogeneity in events (which can be understood as containing event-descriptive data) coming from different sources. In C^{2}offein [19], event sources are encapsulated by means of wrappers. These wrappers map application-specific events into a shared event description composing syntax and structure. [18] propose abstract connectors to hide a set of heterogeneous components, making invisible the fact that different event sources exist.

To integrate data/events from different sources, the approaches mentioned above concentrate on structural aspects, and assume global knowledge by the DBA or the application developer of the assumed semantics of all relevant data and events. This assumption is unrealistic in a large and very dynamic environment like the Internet. Notice that data/events must be compared or correlated externally from the source generating the event. Events containing date or price attributes, require explicit knowledge about modeling assumptions in regard to format or currency to correctly interpret them. Moreover, consumers and producers of events may be previously unknown, therefore a common structural and semantic representation basis (common vocabulary) of the involved events is necessary for their correct interpretation and use [3]. Consider an online auction scenario, where participants may have never met before. Here a common vocabulary is mandatory (normally established using categories of items) and because of its global scope, representations of all the descriptive information require context information, such as, date and time format, metric system, currency, etc. to correctly interpret data.

2.2 Distribution: Detection of Global Composite Events

There are several approaches that deal with distribution, in the area of event propagation [6,14,8,15]. However, they do not consider event composition, where order between events is required to apply event operators (e.g. sequence), or to consume events coming from different locations. Normally, events are timestamped to provide a time-based order, but in open distributed environments global time is not applicable. In [17,25] a global time approximation, known as *2g-precedence*, is proposed. Schwiderski [24] adopted the 2g-precedence model to deal with distributed event ordering and composite event detection. She proposed a distributed event detector based on a global event tree and introduced 2g-precedence based sequence and concurrency operators. However, event consumption is non-deterministic in the case of concurrent or unrelated events. Additionally, the violation of the granularity condition (2g) may lead to the detection of spurious events.

Many projects on event composition in distributed environments, such as [2,22,13,27], either do not consider the possibility of partial event ordering or are based on the 2g-precedence model. Therefore, they suffer from one or more of the following drawbacks: they do not scale to open systems, they provide the possibility of spurious events, and present ambiguous event consumption. In [20] a new approach for timestamping events in large-scale, loosely coupled distributed systems is proposed that uses accuracy intervals with reliable error bounds for timestamping events that reflect the inherent inaccuracy in time measurements .

2.3 Unbundling of Active Database Functionality into Reusable Components

Unbundling is the activity of decomposing systems into a set of reusable components and their relationships [12]. Unbundling active databases consists in separating the active part from active DBMSs and breaking it up into units providing services like, event detection, rule definition, rule management, and execution of ECA rules on one hand and persistence, transaction management and query processing services on the other [11]. A separation of active and conventional database functionality would allow the use of active capabilities depending on given application needs without the overhead of components not needed. Similar approaches were adopted by [7,10,19].

From our point of view, unbundling active functionality from a concrete system and then rebundling the corresponding components to apply them in an open distributed environment is not feasible. Unbundling in this context means to give up the ”closed world” which traditionally underlies a DBMS. Inherent characteristics of open distributed environments impose new requirements that were not considered in centralized environments, like the partial order of events. The consideration of these characteristics has direct impact on the event detector, which is the essential component of an aDBMS [5]. Consequently, it would not be feasible to reuse components taken from centralized DBMSs since they ignore relevant aspects of the new scenario. In addition, the semantics of operators and consumption modes are hard-wired in the code of existing event detectors. Because in the projects mentioned above a generic architecture is defined, there may be difficulties when integrating unbundled and newly developed (autonomous) components. Notice that the meaning of terms employed by different components can conduct to misinterpretations if a common semantic basis, i.e., vocabulary, is not shared.

2.4 Processing ECA-Rules

Rule execution semantics prescribe how an active system behaves once a set of rules has been defined. Rule execution behavior can be quite complex [26,5], but we restrict ourselves to discussing essential aspects. The whole ECA-rule processing process is a sequence of four steps:

1. Complex event detection: event instances generated at event sources feed the complex event detector, which selects and consumes these events to detect specified situations of interest. It binds related event instances with the signaled event.
2. Rule selection: find fireable rules, and apply a conflict resolution policy if needed. There are three basic strategies: a) one rule is selected from the fireable pool, after rule execution the set is determined again, b) sequential execution of all rules in an evaluation cycle, and c) parallel execution of all rules in an evaluation cycle.
3. Condition evaluation: selected rules receive the signaled event instance as a parameter to allow the condition evaluation code to access event information (binding). For some applications it may be useful to delay the evaluation of a triggered rule's condition or the execution of its action until the end of the transaction, or execute them in a separate transaction. These possibilities yield the notion of *coupling modes* [9].
4. Action execution: if the corresponding condition is satisfied context information (signaled event instance) is passed as a parameter to allow the action code to access event information (binding). Transaction dependencies between the evaluation of a rule's condition and the execution of a rule's action are specified using Condition-Action coupling modes.

3 ECA Architecture for Distributed, Heterogeneous Environments

The goal is to provide ECA-rule processing functionality with characteristics similar to a centralized aDBMSs in a distributed component system to support new generation of large scale applications. The active functionality service proposed here is based on a flexible architecture founded on autonomous, combinable and possibly distributed services. Here ontologies play a fundamental role; we use them homogeneously to deal with the integration of events and the interaction of autonomous services. In addition, because our architecture is based on components, ontologies are fundamental for the interaction among components developed independently. The underlying communication between these services is based on a publish/subscribe mechanism, which is suitable for distributed environments and offers other advantages as shown later in this section. In particular, this work puts emphasis on:

– a flexible architecture that can be adapted for different application scenarios,
– the use of an ontology to allow the integration of events coming from heterogeneous sources and also for the infrastructure itself,

– a platform for composition of events coming from heterogeneous sources in distributed environments that deals with partial orderings and the lack of a central clock, and
– providing an active service as composition of other elementary services.

Three main pillars are the basis of our work: an ontology-based infrastructure, event notifications, and a service-based architecture. In the next subsections these three aspects are presented; on this foundation we present our architecture in Section 4.

3.1 Ontology-Based Infrastructure

In our context active functionality mechanisms are fed with events coming from heterogeneous sources. These events encapsulate data, which can only be properly interpreted when sufficient context information about its intended meaning is known. In general this information is left implicit and as a consequence it is lost when data/events are exchanged across institutional or system boundaries. For this reason, to exchange and process events from independent participants in a semantically meaningful way explicit information about its semantics in the form of additional metadata is required.

Our architecture is based on the concepts developed in [4] where shared concepts (ontologies) are expressed through common vocabularies as a basis for interpretation of data and metadata. We represent events, or event content to be precise, using a self-describing data model, called MIX [3]. In the following we refer to events represented based on MIX, i.e. based on concepts from the common ontology, as *semantic events*. MIX refers to concepts from a domain-specific ontology to enable semantically correct interpretation of events, and supports an explicit description of the underlying interpretation context. Simple attributes of an event are represented as triples of the form $SSO = < C, v, \$ >$, with C referring to a concept from the common ontology, v representing the actual data value, and $\$$ providing additional metadata (represented also as MIX objects) to make implicit modeling assumptions explicit. For instance, $SSO = < BidAmount, 99, \{< Currency, "USD" >\} >$.

Complex objects are represented in the form $CSO = < C, \mathbb{A} >$, with C referring to a concept of the ontology, and $\mathbb{A}$ providing the set of simple or complex objects representing its sub-objects. For example, a PlaceBid event can be represented as:

```
CSO = <PlaceBid,   {<ParticipantId,412>,
                    <ItemId,5423>,
                    <BidAmount, 99,{<Currency,"USD">}>,...}>
```

Semantic events from different sources can be integrated by converting them to a common semantic context using conversion functions defined in the ontology [3]. In our work ontologies are used at three different levels: a) the basic level, where elementary ontology functionality and physical representation is defined; b) the infrastructure level, where concepts of the active functionality domain are specified; and c) the domain-specific level, where concepts of the subject domain (e.g. online auctions) are defined.

Basic representation ontology: Here elementary ontology functionality and physical representation concepts like strings, booleans, numbers, lists, etc. are defined.

Infrastructure-specific ontology: All elements related with active functionality are represented with concepts defined here. Difficulties associated with different rule language dialects, ambiguities and imprecise terms are resolved using an explicit common vocabulary. For instance, issues related with the definition of rules like event, condition, action, and so on, are explicitly defined in our infrastructure-specific ontology.

Domain-specific ontology: With the purpose of their integration in mind, events are represented with concepts of a common vocabulary and with context information. Required real world concepts are defined in the domain-specific ontology. Based on this, conversion functions can be applied to integrate data/ events and filters and complex detectors can be defined based on the common ontology without considering peculiarities of event representations and interpretation contexts coming from different sources.

This approach provides the following benefits:

- Independence of the rule definition language: Because there is an explicit definition of terms rule compilers can translate from "any" rule specification to the known target vocabulary. Moreover, rules can be defined directly by applications using an API (accessing the ontology) or by the user in a user-friendly rule language definition.
- Extensibility: New aspects can be incorporated to the rule definition and to the service itself due to the extensibility provided by the underlying ontology support.
- Service interaction "independence": Service interfaces are defined using ontology-based concepts contributing to their clear understanding by service-clients and service-providers.

Furthermore, other aspects related with the infrastructure, in particular, notification-related terms (notification, operational data, detection-time, event source, priority, time-to-live, etc.) are also captured in the infrastructure-specific ontology.

3.2 Events and Notifications

An event is understood here as a happening of interest. Events can be classified as:

- Database events that are further subdivided into data modification and data retrieval events.
- Transaction events refer to the different stages of transaction execution, e.g. begin transaction, commit, rollback, etc.

- Temporal events are classified in absolute, periodic and relative. Absolute temporal events are defined using a particular day and time, while periodic temporal events are signaled repeatedly using time or calendar functions, e.g. every Friday at 11:59PM. Relative ones are defined using a time period with respect to another event, e.g. one week after BeginOfAuction.
- Abstract events or application-defined events are declared by an application, e.g. user-login, auction-canceled. Events of this kind are signaled explicitly by the application.

Observation of happenings include the use of interception and polling mechanisms. In distributed platforms, like CORBA and J2EE, service requests can be intercepted. Using this feature, happenings related with a method execution can be intercepted transparently (without modifying the application). On the other hand, there are event sources that may need to be polled in order to detect events.

The event classification presented above is explicitly specified in the infrastructure-specific ontology. Notice that ontologies in our architecture are extensible. For instance, the representation of a heartbeat[1] is based on the definition of the periodic temporal event, adding in this case some extra information, such as frequency, process identification, etc. Likewise, real-world aspects of a particular domain are represented at the domain-specific layer, e.g. BeginOfAuction as a specialization of an absolute temporal event; ParticipantLogin as application-defined, and so on.

A notification is a message reporting an event to interested consumers. A notification carries not only an event instance but also important operational data, such as reception time, detection time, event source, time to live, priority, etc. As seen on several active system prototypes, complex event detection is mainly based on operational data (particularly comparing timestamps of event instances) while filtering is based on both (i.e. event source and/or attribute values). For this reason, we distinguish the content of a notification into operational data and the event data itself. Operational data concepts are also defined as part of the infrastructure level.

Events coming from different applications are integrated by event adapters. They convert source-specific events into events represented by ontology-based concepts enriched with semantic contexts, i.e. semantic events (see Section 4.2).

3.3 Service-Based ECA-Rule Processing

We realize the ECA-rule processing as a combination of its basic services, i.e. complex event detection, condition evaluation, and action execution service. Event, condition and action services are then combined using a notification service based on a publish/subscribe mechanism which transports event information among them. Subscribers (consumers) place a standing request for events

[1] The heartbeat protocol is based on a message sent between machines at a regular interval with the purpose to monitor the availability of a resource.

by subscribing. On the other hand, a publisher makes information available for its subscribers. Thus, event producers and consumers are decoupled in our architecture. Our ECA-rule service offers the functionality needed to define, remove, activate/deactivate, and search/browse ECA-rules. Figure 1 depicts a configuration of our elementary services using boxes to denote services and lolipops for their interfaces. Circles inside these boxes represent instances of objects under the control of the corresponding service.

Now consider the registration of a rule R1 with the ECA-rule service, as shown in step 1 in Figure 1. Following this is the registration of its corresponding event, condition and action with the proper services (step 2). The complex event detector configures internal objects in order to detect R1's event; then the condition evaluation service instantiates a condition object and subscribes it with R1's event object (step 3). Afterwards, the action execution service instantiates an action object and subscribes it with R1's condition object completing the subscription phase.

Once a triggering event is detected, the complex event detector publishes this happening. That means, all rules that were defined using this triggering event are automatically "activated", in particular their condition objects are notified (step 4a). In this situation, no conflict resolution policy is needed because all rules are executed concurrently (other execution models are possible). When condition objects are notified, they evaluate their predicate and if true, automatically notify the corresponding action objects using the same notification service (step 4b).

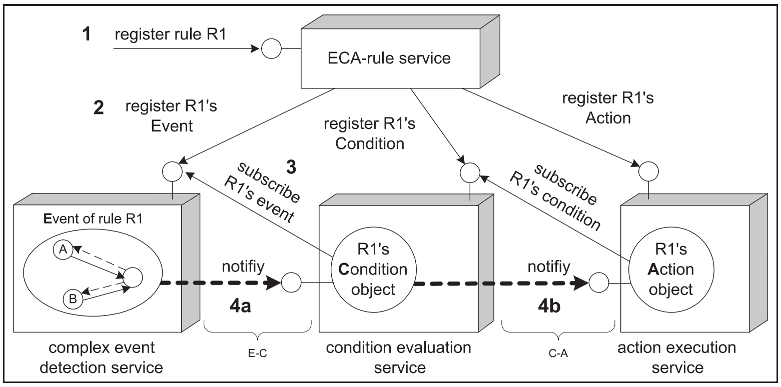


Fig. 1. Interaction among the elementary services (E-C-A)

Services communicate using a notification service. The notifications sent contain a representation of the triggering event, with information about the happening of interest and its context. The communication among these services could be done using other mechanisms, e.g. Remote Procedure Call. However, the publish/subscribe mechanism plays an important role in our architecture providing the following advantages:

– it allows asynchronous communication and decouples event producers and consumers (suitable for open distributed environments),
– it is particularly useful if various rules are associated with the same event,
– it facilitates concurrent rule execution (without centralized rule selection mechanism),
– the notification contains required event information and its context (context propagation),
– it provides a simple and powerful generic communication model,
– it supports location transparency due to subject organization, and
– it helps to explicitly represent dependencies of the flow of work.

Elementary services expose two kinds of generic and very simple interfaces:

– Service interface: defined as a single method that receives as a parameter an event notification which is represented based on the common ontology.
– Configuration interface: for administration purposes, such as register, activate, deactivate, delete, etc.

This simple service interface definition provides flexibility, allowing to configure the flow of service execution easily. For instance, consider the omission of the Condition part in a rule definition (EA-rules). This changes the flow of execution where the Action connects directly to the Event detection. Similarly, event filters can be placed between event source and the complex event detector without requiring to change the code of these components. The destination of the outgoing notifications relies on *concept-based addressing*, which is used for notification dissemination (see Section 4.1).

In addition to the elementary services mentioned above the complex event detection service combines other services which are required for this event detection, like, filtering service, time service, alarm service, event adapters. Furthermore, the following services are used as part of our architecture: ontology service, repository service, and notification service. All these services are described in more detail in the next section.

4 Reference Architecture

Our work is based on a service architecture for extensibility and flexibility reasons. Figure 2 shows, a combination of the basic components to support typical active functionality. Services are implemented using loosely-coupled components to fit distribution requirements. The communication among these services is based on a notification service using a publish/subscribe mechanism. Dashed arrows in Figure 2 depict the use of the underlying notification service.

The services' simple interfaces provide flexibility in order to configure the service execution flow independently of the next service interface in the process chain. In our particular case, services are combined in a chain beginning with the event source and ending with the action execution service. In between optional services, such as event filters, and condition evaluation can be interconnected.

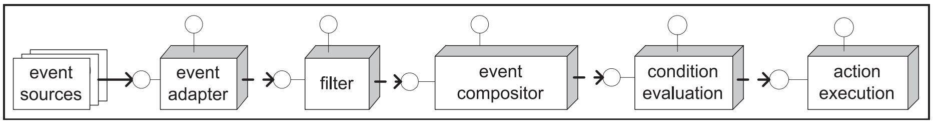


Fig. 2. Basic components to support active functionality

Information about rule definitions, deployment and configuration of their events, conditions and actions, and knowledge about services, is maintained in a repository to support configuration and maintenance. In addition, an ontology service is used to manage concept specifications defined at the three layers of our infrastructure.

4.1 Notification (Event Delivery) Service

A publish/subscribe mechanism decouples producers and consumers of messages. Instead of addressing by location (i.e. by IP number or socket address), publish-subscribe interaction uses the generic communication paradigm of *subject-based addressing*. Producers of messages publish their individual messages under certain subjects. They do this without any knowledge of who or what applications are subscribing to these subjects. Subjects are used to direct messages to their destinations, so applications can communicate without knowing the details of network addresses or connections and the location of message consumers becomes entirely opaque without requiring a name service. Moreover, new message producers and consumers can be introduced at any time. Messages are delivered efficiently to many consumers in an asynchronous way. Message Oriented Middleware (MOM) products and standard specifications like CORBA Notification Service [23] and Java Message Service (JMS) [16] support the publish/subscribe mechanism.

We use X^{2}TS [21] to support coupling modes at the notification service. X^{2}TS integrates notification and transaction services, leveraging multicast enabled MOM for scalable and reliable event dissemination. In addition, it provides the possibility to store messages in a database for event logging.

In our work, we introduced *concept-based addressing*. As its name suggests, subscriptions are made based on the concepts defined in the underlying ontology. Using concept-based addressing consumers subscribe to a concept of interest. This approach is based on the subject-based addressing principles where the subject namespace is hierarchically organized. Here concepts are mapped to the subject namespace, where the concept name is part of the subject, such as PlaceBid. It is also possible that attribute values of a concept constitute part of the namespace in order to allow a more specific subscription, for instance PlaceBid.<ParticipantId>.<ItemId>. In particular, mandatory attributes of a concept are candidates to form part of the namespace. This way, the destination of notifications is determined by self-contained information. Namespace organization is maintained in the repository.

Because concepts are represented using a common ontology, consumers do not need to take care of proprietary representations. Moreover, if concepts are extended nothing must be changed at the consumer side.

4.2 Event Sources and Adapters

An API to describe event context and to signal semantic events is provided. Based on this API event adapters convert source events into concepts based on the domain-specific vocabulary adding proper context information to support its correct interpretation. Examples of event adapters are:

– *XML adapters* that translate XML-based messages into semantic events.
– *Database adapters* that are used to signal database operations outside the database. With this purpose, trigger mechanisms can be used to detect the event and then stored procedures are used to generate the corresponding semantic event.
– *Interceptors* that are used to intercept a service request (before or after execution). Once a request is intercepted a corresponding event is signaled.
– *Alarm adapters* that play the role of a mediator to scheduled time-related events (using the alarm service) and when proper generates time-related events based on the ontology as a result.
– *E-mail adapters* that intercept e-mails according to specified e-mail properties, like sender address, subject, etc. Once caught, the necessary information is extracted from an e-mail and converted into a semantic event.

4.3 Alarm Service

This service is also considered as a source of temporal events (absolute, relative and periodic). These events require a clock scheduler in order to signal scheduled temporal events. For instance, an absolute temporal event indicating BeginOfAuction, can be defined as "February 19, 2001 at 9:00AM GMT-5" which means that at the specified time it must be signaled. This service can also be useful for the infrastructure itself, e.g. in a distributed environment where a heartbeat mechanism is needed to produce (and consume) heartbeats within a determined periodicity. It is important to notice that in this kind of scenario, clocks used by these services must be synchronized with the timestamp service explained below.

4.4 Time (Timestamp) Service

Timestamps allow events to be ordered. This order takes a fundamental role in event consumption (e.g. chronicle) and when using time-related event operators. Depending on the given system environment, e.g. distributed or centralized, different timestamp models can be used. For example, for centralized environments a simple timestamp mechanism may be sufficient since only one clock is used to timestamp events. For distributed closed networks the 2g-precedence model may fit. For open distributed environments the accuracy interval model [20] can be used. The implementation of a timestamp service should explicitly declare all assumptions made. For this reason, a timestamp ontology is required to describe semantic assumptions about the timestamp mechanism like observation (detection time, occurrence time, reception time), clock source (local, remote, local synchronized), clock granularity, etc. The timestamp ontology is important to determine the compatibility of timestamp mechanisms.

4.5 Filter Service

Filters select notifications by discarding events based on predicates defined on event attributes. For instance, events coming from a particular source, or events which contain prices over US$ 100 are not of interest so they can be filtered and do not reach the consumer. Because events and notifications are represented using concepts from the common ontology, event filters can be specified at a domain-specific level, and are independent from source-specific representations.

4.6 Complex Event Detection

This service must be configured to recognize complex situations based on primitive and composite events. Situations of interest are described using event operators. There are several aspects that are involved when recognizing complex event situations such as those summarized in [1,28]. In particular, the inherent characteristics imposed by large-scale distributed environments have to be considered: the partial order of events, transmission delays, and possible failures at the sources or in the communication channel. The basic infrastructure of the complex event detector service is based on the idea of components and containers. Components are the event operators (also named compositors here) that are plugged into the container. The container itself is the complex event detector kernel which controls the event detection process. This approach avoids hard-wired event operators within the complex event detector, plugging-in only the set of operators related with the specified rules, and it provides a flexible framework to define other event operators. Compositors subscribe to primitive events or other compositors according to the complex event definition. Event instances are propagated to subscribers using the notification service described in Section 4.1. Compositors can also be configured in order to apply a consumption policy of event instances (consumption mode). This depends on the implementation of basic methods that allow to order events based on the ontology-based timestamp mechanism.

4.7 Condition and Action Service

The condition service provides an interface to register the condition (predicate) of a particular rule that must be evaluated once the corresponding event is signaled. This service plays the role of a factory, instantiating a condition object, setting its pertinent properties (predicate to evaluate, drivers to allow predicate evaluation, etc.), and subscribing it to its corresponding triggering event taking into account the specified coupling mode between event and condition. Notice that different conflict resolution policies can be applied here but concurrent rule execution is assumed in order to improve scalability and performance.

Likewise, the action service provides an interface to register the action of a rule, instantiating an action object, setting its properties to receive the corresponding notification. Depending on the rule definition, condition and action objects are associated with objects that provide the means to: access databases, invoke methods on distributed objects or wrapped legacy applications, invoke transaction control operations, access messaging services (e-mail, queues, SMS, fax), etc.

4.8 Repository and Ontology Service

We use an ontology server to store and manage the common vocabulary used in our framework. This vocabulary provides the extensible domain- and infrastructure-specific description basis to which all other services refer. The ontology server provides a common access point to the vocabulary, and provides concept definitions and textual, human-readable descriptions of available concepts for interactive exploration by developers and end users. In our repository are stored: rule definitions, their deployment and configuration, event constellations, configuration of adapters, service configurations, and subject namespace organization. This repository is used for configuration and maintenance purposes.

5 Conclusions and Future Work

We presented a service-based architecture to support active functionality which is based on a common vocabulary (ontology), services, and event notifications. Ontologies are used as a common interpretation basis to enable semantically correct interpretation of events and notifications in open heterogeneous environments. Our ontology-based infrastructure applies homogeneously the ontology approach not only to integrate events from different sources but also to support the integration among elementary services. This allows filters and operators on events to be defined at a higher-level without taking care of source-specific representation peculiarities. ECA-rule processing in our architecture is decomposed into elementary services. These services provide a very simple and generic interface, where parameters of methods are represented using the common ontology. Therefore, the flow of work through services can be easily configured – omission or inclusion of services like condition evaluation, event filtering or complex event detection is made easy. Notifications are used to carry events from their source to interested consumers (services), i.e. notifications are the means for services to interact. For this purpose, a notification service, based on a publish/subscribe mechanism using concept-based addressing, is used. In addition, this service supports event storage and different coupling modes. Because of this conceptual foundation, our architecture promotes flexibility, extensibility and integration for large-scale Internet-based applications.

We use the Java language to specify ontology concepts and their relationships, thus avoiding any impedance mismatch between programming language and ontology specification language, and allowing the shipping of ontology concepts between different platforms without further transformations. In addition to data portability, Java supports code portability which is an important issue here since rule enforcement may be necessary to run at different tiers and at different run-time configurations. For instance, to control business rules at the client, at the middle tier or at the server. Ontology support is completely implemented and many of the ontology concepts are already defined. Our implementation also includes a running notification service that supports transactions and coupling modes, and some event adapters like XML adapter, alarm, and application adapter.

Current research involves issues related with complex event detection in open distributed environments. In particular three issues are being investigated. First, consumption modes which should take into account partial order of events and how to cope with uncertainty of event order. Second, we are looking for a minimalistic set of (low-level) event operators, and based on them define domain-specific powerful (high-level) event operators. Finally, the extension of the timestamp ontology is required in order to correctly interpret and compare timestamps coming from distributed sources. Therefore, different time synchronization dimensions and event observation mechanisms must be studied and properly organized and represented. Another aspect to be studied, is the validation of different service configurations to avoid unimplementable or inconsistent services.

References

1. (ACT-NET) K. Dittrich, S. Gatziu, A. Geppert. The Active Database Management System Manifesto: A Rulebase of ADBMS Features. In Proc. of RIDS, LNCS 985, Sept 1995. 207
2. J. Bacon, K. Moody, J. Bates. Active Systems. Technical Report. Computer Laboratory, University of Cambridge, Dec 1998. 198
3. C. Bornhövd, A. Buchmann. A Prototype for Metadata-Based Integration of Internet Sources. In Proc. Intl Conf. on Advanced Information Systems Engineering, 1999. 197, 200
4. C. Bornhövd. Semantic Metadata for the Integration of Heterogeneous Internet Data (in German). Ph.D. Thesis, Department of Computer Science, Darmstadt University of Technology, Germany, 2000. 200
5. A. Buchmann. Architecture of Active Database Systems. In Norman Paton (editor), Active Rules in Database Systems, Springer, 1999. 198, 199
6. A. Carzaniga. Architectures for an Event Notification Service Scalable to Wide-area Networks, Ph.D. Thesis, Politecnico di Milano, Italy, 1998. 197
7. C. Collet, G. Vargas-Solar, H. Grazziotin-Ribeiro. Towards a Semantic Event Service for distributed Active Database Applications. DEXA'98, LNCS 1460, September 1998. 198

8. G. Cugola, E. Di Nitto, A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In Proc. Intl. Conf. on Software Engineering (ICSE), 1998. 197
9. U. Dayal, et. al The HiPAC Project: Combining Active Databases and Timing Constraints. SIGMOD Record 17(1), 1988. 199
10. H. Fritschi, S. Gatziu, K. Dittrich. FRAMBOISE - an Approach to construct Active Database Mechanisms. Tech. Rep. IFI-97-04, Inst. für Informatik, Univ. of Zurich. 1997. 198
11. S. Gatziu, A. Koschel, G. von Bültzingsloewen, H. Fritschi. Unbundling active functionality. SIGMOD Record, 27(1), 1998. 196, 198
12. A. Geppert, K. Dittrich. Bundling: A new Construction Paradigm for Persistent Systems. Networking and Information Systems Journal, 1(1), June 1998. 196, 198
13. A. Geppert, D. Tombros. Event-based Distributed Workflow Execution with EVE. In Proc. of Middleware '98, Sept 1998. 198
14. R. Gruber, B. Krishnamurthy, E. Panago. The Architecture of the READY Event Notification Service. In Proc. Workshop on Distributed Computing Systems Middleware, 1999. 197
15. A. Hinze, D. Faensen. A Unified Model of Internet Scale Alerting Services. In Proc. Intl. Computer Science Conference (ICSC), LNCS 1794, 1999. 197
16. JavaSoft, Java Message Service (JMS) spec. 1.0.1, Oct., 1998. 205
17. H. Kopetz. Sparse Time versus Dense Time in Distributed Real-Time Systems. In Proc. Intl. Conf. on Distributed Computing Systems (ICDCS), 1992. 198
18. A. Koschel, S. Gatziu, G. von Bültzingsloewen, H. Fritschi. Unbundling active functionality. In A. Dogac, et. al (editors), Current Trends in Data Management Technology, IDEA Group Publishing, 1999. 196, 197
19. A. Koschel, P. Lockemann. Distributed Events in Active Database Systems - Letting the Genie out of the Bottle. Journal of Data and Knowledge Engineering, Vol. 25, 1998. 197, 198
20. C. Liebig, M. Cilia, A. Buchmann. Event Composition in Time-dependent Distributed Systems. In Proc. Intl Conf. on Cooperative Information Systems (CoopIS), 1999. 198, 207
21. C. Liebig, M. Malva, A. Buchmann. Integrating Notifications and Transactions: Concepts and X2TS Prototype. In Intl Workshop on Engineering Distributed Objects (EDO), 2000. 205
22. C. Ma, J. Bacon. COBEA: A CORBA-Based Event Architecture. In Proc. of the USENIX Conf. on Object-Oriented Technologies and Systems, June 1998. 198
23. Object Management Group (OMG), Notification Service Specification. Technical Report telecom/98-06-15, May, 1998. 205
24. S. Schwiderski. Monitoring the Behavior of Distributed Systems, Ph.D. Thesis, Selwyn College, Computer Lab, University of Cambridge, June 1996. 198
25. P. Verissimo. Real-Time Communication. In Sape Mullender (Editor), Distributed Systems, Addison-Wesley, 1993. 198
26. J. Widom, S. Ceri (editors), Active Database Systems: Triggers and Rules for Advanced Database Processing. Morgan Kaufmann, San Francisco, CA, 1996. 199
27. S. Yang, S. Chakravarthy. Formal Semantics of Composite Events for Distributed Environments. In Proc. Intl. Conf. on Data Engineering (ICDE), 1999. 198

28. D. Zimmer, R. Unland. On the Semantics of Complex Events in Active Database Management Systems. In Proc. Intl. Conf. on Data Engineering (ICDE), 1999. 207

Generic Constraints for Content-Based Publish/Subscribe

Gero Mühl*
Darmstadt University of Technology
Wilhelminenstr. 7, D-64283 Darmstadt, Germany
gmuehl@gkec.tu-darmstadt.de

Abstract. Publish/subscribe interaction enables the loosely coupled exchange of asynchronous notifications. Matching notifications to subscriptions and routing notifications from producers to interested consumers are the main problems in large-scale publish/subscribe systems. Content-based selection provides great flexibility because it allows to categorize notifications with respect to multiple dimensions but it also requires complex routing strategies. Previous work has tightly coupled content-based routing and matching algorithms to the constraints that can be used to select notifications of interest. This paper introduces the idea of generic constraints for content-based selection that are separated from the underlying matching and routing algorithms. Therefore, new constraints can be added easily. Moreover, this paper introduces the idea of filter merging that can improve the scalability of a notification service. Finally, we present novel algorithms for matching, covering, and merging.

1 Introduction

Today, the web is dominated by client initiated request/reply. HTTP is a protocol that is based solely on this model. This interaction scheme is appropriate for spontaneous catalog browsing and similar applications but on the other hand it seriously limits the type of applications that can be realized efficiently.

Applications that need a consistent and up-to-date view on continuously changing information published on the web spend an inordinate amount of time polling known sites for updates [13]. This approach leads to chained inaccuracies and seriously limits scalability. Asynchronous notifications are needed as additional mechanism to enable the proactive delivery of information updates. Moreover, request/reply tightly couples information consumers to the information providers. Decoupling of information producers and consumers removes explicit dependencies and therefore facilitates extensibility and flexibility.

Publish/subscribe offers loosely coupled exchange of asynchronous notifications. Although publish/subscribe messaging can also be used for request/reply, we are mainly interested in supporting event-based applications. An event-based system consists of producers and consumers. Producers publish events and consumers subscribe to the set

* The author's work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the PhD Program "Enabling Technology for Electronic Commerce"

© Springer-Verlag Berlin Heidelberg 2001

of events they are interested in. The underlying event notification service is responsible for delivering each published event to all interested consumers. To use a publish/subscribe messaging middleware does not necessarily imply an event-based style: the event-based style additionally assumes that a publisher has no direct expectation on the receivers of a notification. A broad range of applications can benefit from an event-based realization. For example, electronic trading and auction platforms are inherently event-based [4].

The expressiveness of the subscription model is crucial for the flexibility and the scalability of a notification service. Insufficient expressiveness can lead to unnecessary broad subscriptions saturating the network and raising the need for additional client-side filtering. In the worst case, when interests cannot be mapped to selection mechanisms all events must be delivered. On the other hand, scalable implementations of more expressive subscription models require complex delivery strategies.

The first generation of publish/subscribe systems has used *channels* [18,25]. A notification is published to a specific channel specified by the publisher and delivered to all consumers that have subscribed to this channel. Channels allow for efficient delivery but their expressiveness is rather limited leading to inflexible applications.

Subject-based publish/subscribe systems [20, 26, 28] associate with each notification a single subject that is usually specified as a string. Subjects can be arranged in a tree by using a dot notation, and therefore, they enable more complex matching than channels. Subjects are rather inflexible because changes to the subject tree can require major application fixes. In any case, the question arises: "Who defines the subject tree?". Subjects are suitable to divide the notification space with respect to one dimension. The use of several dimensions leads to an explosion of the tree size because of subtree repetition.

The *content-based* approach first mentioned in [5] allows to use the whole content of a notification for filtering. Clearly, it is the most flexible mechanism but on the other hand it requires the most complex infrastructure. Content-based mechanisms are used by a set of notification services including Elvin [23], Gryphon [1, 2, 21], Keryx [30], Siena [6, 7], Le Subscribe [11, 22], Jedi [9], the CORBA Notification Service [19], and Rebeca [12]. They have also been proposed for inter-agent communication [24] and for defining continual queries [16].

One of the main problems in large-scale content-based publish/subscribe systems is matching notifications to subscriptions and routing notifications from producers to interested consumers. Clearly, a centralized solution relying on a single notification broker is neither scalable nor fault-tolerant. The alternative is to use a set of distributed brokers in which each broker has a set of local clients and is additionally communicating with some of the other brokers. In that case, notifications are propagated from producers to consumers along a path of interconnected brokers. Most work either floods notifications to the broker network or, if filtering is performed at intermediate brokers, it is assumed that each broker has global knowledge about all active subscriptions. This knowledge is constructed by flooding subscriptions into the network. It enables each broker to decide to which brokers it must forward incoming notifications. Therefore, subscriptions are forwarded to ensure the delivery of all matching notifications.

As an alternative Rosenblum, Carzaniga, and Wolf [6, 7] have presented the content-based routing algorithms of Siena. They reduce the amount of knowledge needed by a broker to make notification forwarding decisions by applying selective filter forwarding (i.e. subscription and advertisement forwarding). Their routing algorithms are based on covering relations among filters. For example, a subscription does not need to be forwarded by a broker to another broker if it has already forwarded a subscription that covers the former because in that case already all matching notifications are delivered. Unfortunately, an algorithm to determine covering relations is not presented by the authors. In Siena notifications consist of a set of attributes which are name value pairs whose values are instances of some predefined primitive types (e.g. integers). New types cannot be introduced by the application programmer.

In this paper, we present the concept of generic content-based constraints that are clearly separated from the matching and routing algorithms. In consequence, new constraints and value types can be easily integrated. Examples are described that cover a wide range of practically important constraints and illustrate the feasibility of our approach. We also introduce a mechanism to merge filters that further improves filter forwarding and therefore the scalability of a notification service. Moreover, we present a set of algorithms (matching, covering, and merging) which incorporate our concepts. Finally, we describe the implementation of our notification service. It uses routing algorithms similar to those of Siena but extended by support for merging. Due to space limitations we do not discuss details of routing algorithms in this paper.

The remainder of this paper is organized as follows: Section 2 describes in detail the content-based data model that underlies this paper. In the third section we present our approach to supporting generic constraints and types. Section 4 introduces the concept of filter merging and in section 5 algorithms for matching, covering, and merging are presented that support the concept of generic constraints. Section 6 discusses related work and section 7 outlines our implementation.

2 Content-Based Data Model

2.1 Notifications

A *notification* is a message that contains information about an event that has occurred. Formally, a notification *n* is a set of *attributes* $\{A_1, \ldots, A_n\}$ where each A_i is a *name value pair* (n_i, v_i) with name n_i and value v_i. We assume that names are unique, i.e. $i \neq j \Rightarrow n_i \neq n_j$, and that there exists a function that uniquely maps each n_i to a type T_j that is the type of the corresponding value v_i.

We distinguish between *simple values* that are a single element of the domain of T_j, i.e. $v_i \in dom(T_j)$, and *multi values* that are a finite subset of the domain, i.e. $v_i \subseteq dom(T_j)$. An example of a simple notification is {(*type*, *StockQuote*), (*name*, "*Infineon*"), (*price*, *45.0*)}.

2.2 Filters

A *filter* F is a stateless boolean function that is applied to a notification, i.e. $F(n) \rightarrow \{true, false\}$. A notification *matches* F if $F(n)$ evaluates to true. Consequently, the set of matching notifications $N(F)$ is defined as $\{n | F(n) = true\}$. Two filters F_1, F_2 are *identical*, written $F_1 \equiv F_2$, if and only if $N(F_1) = N(F_2)$. Moreover, they are *overlapping* if $N(F_1) \cap N(F_2) \neq \emptyset$ and otherwise they are *disjoint*.

Filters are used to define subscriptions and advertisements (see Sect. 2.4). Note that in contrast to filters, statefull boolean functions are known as *pattern recognizer*. They are outside the scope of this paper.

A filter is usually given as a boolean expression that consists of predicates that are combined by boolean operators (e.g. *and*, *or*, *not*). We call a filter consisting of a single atomic predicate as *simple filter* or *constraint*. Filters that are derived from simple filters by combining them with boolean operators are *compound filters*. A compound filter that is a conjunction of simple filters is called a *conjunctive filter*. It is sufficient to consider conjunctive filters because a compound filter can always be broken up into a set of conjunctive filters that are interpreted disjunctively and can be handled independently.

2.3 Attribute Filters

In the following we restrict a filter F to be a conjunctive filter whose predicates are *attribute filters*, i.e. $F = AF_1 \wedge \ldots \wedge AF_k$. An attribute filter depends on a single attribute, i.e. its value. It is defined as a tuple $AF_i = (n_i, Op_i, C_i)$ where n_i is an attribute name, Op_i is a test operator and $C_i = \{c_{i1}, \ldots, c_{il}\}$ is a set of constants. The name n_i determines to which attribute the constraint applies. If the notification does not contain an attribute with name n_i then AF_i evaluates to false. Otherwise, the operator Op_i is evaluated using the value v_j of the addressed attribute and the specified set of constants C_i. The outcome of AF_i is defined as the result of Op_i that evaluates either to true or false. Therefore, a notification n matches F if it satisfies all attribute filters. An example for a conjunctive filter consisting of attribute filters is {(*type*=*StockQuote*), (*name*= "*Infineon*"), ($price \notin$ [30,40])}.

To enable efficient evaluation of covering and merging we allow at most one attribute filter per attribute. Space limitations prevent us from discussing the reason for this restriction in more detail, but it enables us to reduce covering and merging of filters to their respective attribute filters (see Sect. 2.5). In contrast to most other work (e.g. Siena) we support constraints that depend on more than one constant. This enables more operators and enhances the expressiveness of the filtering model. This can be done without affecting scalability as we will see in later sections.

2.4 Subscriptions and Advertisement

A *subscription S* is a filter that is issued by a consumer to indicate its interest to receive future notifications. The subscription indicates that all and exclusively those notifications that are matched by *S* should be delivered to the consumer. Therefore, a consumer should never receive a notification that is not matched by at least one of its active subscriptions.

Advertisements are filters that are issued by producers to indicate their intention to publish future notifications. All notifications that are published by a particular producer must be matched by one of the advertisements that it has issued. The details of the semantics of subscriptions and advertisements are essential, in particular for achieving optimization. They are used by the notification service to route notifications from publishers to subscribers.

2.5 Covering Relations

Similar to Siena we introduce a *covering relation* between two filters: F_1 covers F_2, also written $F_1 \supseteq F_2$, if and only if $N(F_1) \supseteq N(F_2)$. The covering relation is transitive and induces a partial order over a given set of filters. It is easy to see that if $F_1 \supseteq F_2$ then $n \notin N(F_1)$ implies $n \notin N(F_2)$ and $n \in N(F_2)$ implies $n \in N(F_1)$. Moreover, $F_1 \supseteq F_2 \wedge F_1 \subseteq F_2$ is equivalent to $F_1 \equiv F_2$.

In our case (i.e. conjunctive filters with at most one attribute filter imposed on a single attribute) the covering problem of filters can be reduced to the covering problem of attribute filters. A filter F_1 covers a filter F_2 if and only if for each attribute filter AF_1^i of F_1 an attribute filter AF_2^j of F_2 exists that is covered by the former i.e.:

$$(\forall i \exists j . AF_1^i \supseteq AF_2^j) \Leftrightarrow F_1 \supseteq F_2 \qquad (2.1)$$

Covering relations among filters are exploited by the routing algorithm to reduce the number of subscriptions that must be forwarded. Eq. 2.1 can be directly transformed to an algorithm that detects covering and identity among filters. However, in section 5 we present an algorithm that is more efficient than this naive algorithm. But before we do this, we first discuss our generic approach and introduce filter merging.

3 Generic Constraints and Types

Former work dealing with content-based notification selection mechanisms often tightly integrated the constraints that can be put on values and the types of values supported with the matching and routing algorithms [1, 2]. An exception is Siena where matching and routing algorithms are separated from constraints. However, they only support a fixed set of constraints on some predefined primitive types.

We propose to use a collection of abstract attribute filter classes. Each of these classes offers a generic implementation of the methods needed by the matching and routing algorithms (e.g the covering and matching test) and imposes a certain type of constraint on an attribute that can be used with values of all types that implement the operators needed. The appropriate implementation of the operators are called by the constraint class at runtime using polymorphism. This enables to define and support new constraints and types without requiring changes to the routing and matching algorithms. Note, that although we propose an object-oriented approach it is not mandatory to use it.

For example, we have implemented a constraint class that implements comparison constraints on totally ordered sets (see Sec. 3.4). This class can be used to impose comparison constraints on all kinds of ordered values (e.g. integer numbers). Consider a type “person” that consists of first and second name, the date of birth, and the place of birth. This type is easily supported by providing implementations for the comparison operators which are called by the constraint class to provide the covering and matching methods using polymorphism.

In the next subsections, some generic attribute constraints are presented that cover a wide range of practically relevant constraints, but more importantly, they illustrate the feasibility of our approach. Of course this collection is not exhaustive but other constraints can be integrated easily. For example, intervals could be used as values. In this case the same operators as defined in Sect. 3.8 can be used because intervals are essentially sets. In our view, the investigation of a subset of regular expressions is also promising. The meaning of a single row in the Tables 1 through 6 that are presented in the next subsections is:Given AF_1 and AF_2 as specified in column 1 and 2 $AF_1 \supseteq AF_2$ if and only if the condition in the column 3 is satisfied. In order to test whether a filter covers another, covering must hold for all attributes.

3.1 General Constraints

We consider two general constraints that can be imposed on all attributes regardless of the type of their value: *exists(n)* tests whether an attribute with name *n* is contained in a given notification, i.e. whether $\exists A_i.n_i = n$. The *notExists(n)* constraint is the negation of the *exists(n)* constraint. The *exists* constraint covers all other constraints that can be imposed on an attribute except the *notExists* constraint. The *notExists* constraint on the other hand only covers itself.

3.2 Constraints on the Type of Notifications

Most work on notification services has a notion of types or classes of notifications. Usually, the type of a notification is specified by a textual string that can be tested for equality and prefix. If a dot notation is used this enables the support of a type hierarchy with single inheritance. Multiple inheritance cannot be supported this way.

We argue for direct support of notification types in order to enable multiple inheritance, automatic propagation of interest in sub-classes [3], and to achieve a better programming language integration [10]. Moreover, type inclusion tests can be evaluated more efficiently than the corresponding string operation (i.e. whether the string starts with a given prefix) [29]. Naturally, we assume that the set of attributes that can be contained in a notification of type *T* is a superset of the union of all attribute names of all supertypes of *T*.

Consequently, we introduce a separate constraint that evaluates to true if *n* is an instance of type *T* and false otherwise, written *n instanceof T*. A constraint *n instanceof* T_1 covers a constraint *n instanceof* T_2 if and only if either T_1 is the same type as T_2 or T_1 is a supertype of T_2.

3.3 Equality and Inequality Constraints on Simple Values

The simplest constraints that can be imposed on a value are tests for equality and inequality. Covering implications among these tests can always be reduced to a simple comparison of their respective constants (see Table 1).

Table 1. Covering implications among equality and inequality constraints on simple values

AF_1	AF_2	$AF_1 \supseteq AF_2$ iff
$x = c_1$	$x = c_2$	$c_1 = c_2$
$x \neq c_1$	$x = c_2$ $x \neq c_2$	$c_1 \neq c_2$ $c_1 = c_2$

3.4 Comparison Constraints on Simple Values

Another common class of constraints are comparisons on values for which the domain and the comparison operators define a totally ordered set (e.g. integers with the usual comparison operators). Again, covering among these tests can be reduced to a simple comparison of their respective constants. Table 2 depicts covering implications of inequality and greater than, for brevity the other comparison operators are left out.

Table 2. Covering implications among comparison constraint on simple values

AF_1	AF_2	$AF_1 \supseteq AF_2$ iff
$x \neq c_1$	$x < c_2$	$c_1 \geq c_2$
	$x \leq c_2$	$c_1 > c_2$
	$x = c_2$	$c_1 \neq c_2$
	$x \geq c_2$	$c_1 < c_2$
	$x > c_2$	$c_1 \leq c_2$
$x > c_1$	$x = c_2$	$c_1 < c_2$
	$x > c_2$	$c_1 \leq c_2$
	$x \geq c_2$	$c_1 < c_2$

3.5 Interval Constraints on Simple Values

Interval constraints allow to test whether a value *x* is within a given interval *I* or not, i.e. $x \in I$ and $x \notin I$ respectively, where *I* is a closed interval $[c_1, c_2]$ with $c_1 \leq c_2$. Here, computing coverages involves two comparisons (see Table 3).

Table 3. Covering implication among interval constraints on simple values

AF_1	AF_2	$AF_1 \supseteq AF_2$ iff
$x \in I_1$	$x \in I_2$	$I_1 \supseteq I_2$
$x \notin I_1$	$x \notin I_2$	$I_1 \subseteq I_2$

3.6 Constraints on Strings

In addition to the comparison operators based on the lexical order, we define a prefix, a substring, and a postfix operator. Constraints on strings can be used to realize subjects. Computing coverages among them requires a single test (see Table 4).

Table 4. Covering implication among constraints on strings

AF_1	AF_2	$AF_1 \supseteq AF_2$ iff
s prefix S_1	s prefix S_2	S_1 prefix S_2
s postfix S_1	s postfix S_2	S_2 postfix S_1
s substring S_1	s substring S_2	S_1 substring S_2

3.7 Set Constraints on Simple Values

Set constraints on simple values test whether a value is or is not a member of a given set. In order to compute a coverage among two of these constraints a single set inclusion test is sufficient (see Table 5). Its complexity depends on the characteristics of the underlying set. Set constraints can also be combined with comparison constraints if the domain of the value is a totally ordered set. These covering implication are left out because of space limitations.

Table 5. Covering implications among set constraints on simple values

AF_1	AF_2	$AF_1 \supseteq AF_2$ iff
$x \in M_1$	$x \in M_2$	$M_1 \supseteq M_2$
$x \notin M_1$	$x \notin M_2$	$M_1 \subseteq M_2$

3.8 Set Constraints on Multi Values

The idea of multi values is to allow a value to be a set of elements. This enables set-oriented operators which are defined on a multi value $X = \{v_1, \ldots, v_n\}$. For example, the following common operators can be defined:

X subset M $\Leftrightarrow X \subseteq M$,	X superset M $\Leftrightarrow X \supseteq M$,
X contains $a_1 \Leftrightarrow a_1 \in X$,	X notContains $a_1 \Leftrightarrow a_1 \notin X$,
X disjunct M $\Leftrightarrow X \cap M = \emptyset$,	X overlaps M $\Leftrightarrow X \cap M \neq \emptyset$.

To determine covering with respect to these constraints either the evaluation of a set inclusion test or of a set membership test is needed (see Table 6).

Table 6. Covering implications among set constraints on multi values

AF_1	AF_2	$AF_1 \supseteq AF_2$ iff
X subset M_1	*X subset* M_2	M_1 *superset* M_2
X contains a_1	*X superset* M_2	$a_1 \in M_2$
X superset M_1	*X superset* M_2	M_1 *subset* M_2
X notContains a_1	*X disjunct* M_2	$a_1 \in M_2$
X disjunct M_1	*X disjunct* M_2	M_1 *subset* M_2
X overlaps M_1	*X overlaps* M_2	M_1 *superset* M_2

4 Merging of Filters

In this section, we introduce a mechanism to merge filters. Merging can be used in addition to covering to minimize the knowledge necessary for notification forwarding decisions. For example, a single subscription can substitute a set of subscriptions if it covers them.

In order to define the merging of filters we generalize the covering relation introduced above. A filter F covers a set of filters $\{F_1, \ldots, F_n\}$ if and only if $N(F) \supseteq N(F_1) \cup \ldots \cup N(F_n)$. A covering is said to be *perfec*t if the equality holds, otherwise it is called *imperfect*. In the latter case, $N(F)$ is a real superset of $N(F) \supseteq N(F_1) \cup \ldots \cup N(F_n)$. If F is an imperfect merger of $\{F_1, \ldots, F_n\}$ then $n \in N(F_1) \vee \ldots \vee n \in N(F_n)$ implies $n \in F$. If *F* is a perfect merger, this implication can be strengthened to an equivalence.

The aim of merging is to determine a filter *F* that is a perfect merger of a set of filters. Merging of filters can be used to drastically reduce the amount of subscriptions and advertisements that have to be stored by the routing algorithm. Perfect merging is clearly preferable, although it cannot always be achieved.

In our case, where a filter is a conjunction of attribute filters with at most one attribute filter per attribute, the merging problem can be reduced to the merging of attribute filters. A set of filters $\{F_1, \ldots, F_n\}$ can be merged perfectly to a single filter *F* if the constraints they impose on all but one attribute are identical and the attribute filters of the distinctive attribute filter can be merged. For example, a set of attribute filters imposed on the same attribute with name *n* can be merged to an *exists(n)* test if for all values v_i at least one of them is satisfied.

4.1 Perfect Merging

Merging is implemented by each generic attribute filter class. Table 7 below presents some versatile perfect merging rules. The meaning of a single row is that AF_1 and AF_2 can be perfectly merged to AF if the merging condition holds.

Table 7. Some versatile perfect merging rules for attribute filters

AF_1	AF_2	Merging Condition	$AF = AF_1 \cup AF_2$
$x \in M_1$	$x \in M_2$	-	$x \in M_1 \cup M_2$
$x \notin M_1$	$x \notin M_2$	$M_1 \cap M_2 = \emptyset$ $M_1 \cap M_2 \neq \emptyset$	$\exists x$ $x \notin M_1 \cap M_2$
X overlaps M_1	*X overlaps* M_2	-	*X overlaps* $M_1 \cup M_2$
X disjunct M_1	*X disjunct* M_2	$M_1 \cap M_2 = \emptyset$ $M_1 \cap M_2 \neq \emptyset$	$\exists X$ *X disjunct* $M_1 \cap M_2$
$x = a_1$	$x \neq a_2$	$a_1 = a_2$	$\exists x$
$x < a_1$	$x > a_2$ $x \geq a_2$	$a_1 > a_2$ $a_1 \geq a_2$	$\exists x$
$x \leq a_1$	$x > a_2$ $x \geq a_2$	$a_1 \geq a_2$	$\exists x$

The first two rules can also be applied to equality and inequality tests because $x = a_1 \Leftrightarrow x \in \{a_1\}$ and $x \neq a_1 \Leftrightarrow x \notin \{a_1\}$.

```
Matching Algorithm
Input:   notification n, set of filters F
Output:  the set M of all filters in F that match n.
{
   <For each filter in F a counter is initialized to zero.>
   for <each A_i contained in n> {
      for <each filter S in F that has a constraint on A_i that is
             satisfied by the value of the corresponding attribute of n> {
         <Increment the counter of S>
      };
   };
   for <each A_i that is not contained in n but that could be contained> {
      for <each Filter S in F that has a notExists constraints on A_i> {
         <Increment the counter of S>;
      };
   };
   M:=<all filters in F whose counter is equal to their number of attribute
           filters>
}
```

Fig. 1. Matching algorithm based on counting satisfied attribute filters

4.2 Imperfect Merging

At a first glance, imperfect merging seems to be less promising but in situation in which perfect merging is either too complex or not computable it might be a good compromise. Clearly, there exists a trade-off between filtering overhead and network resource consumption. Imperfect merging may result in notifications being forwarded that do not match any of the original subscriptions, but on the other hand, it reduces the amount of subscriptions and advertisement that must be dealt with.

For example, filters that differ in few attribute filters could be merged imperfectly by imposing on each attribute a constraint that covers all original constraints. Statistical online evaluation of filter selectivity could be used as basis for merging decisions in order to enable adaptive filtering strategies. Imperfect merging requires further investigation.

5 Algorithms for Matching, Covering, and Merging

In this section, algorithms for matching and covering are presented that are superior to the naive algorithms. Moreover, we describe an algorithm for merging of filters. The algorithms exploit our generic approach: Each generic constraint class (e.g. constraints on ordered values) offers specialized indexing data structures to manage constraints on attributes. For example, hashing is used for equality tests.

Some matching algorithms require a costly compilation step (e.g. [14]) that makes them less suitable for dynamically changing systems. In contrast our algorithms allow to add or remove filters at any time although of course the insertion/deletion of a filter causes some overhead.

```
Covering Algorithm
Input:   filter $F_1$, set of filters $F$
Output:  the set $C$ of all filters in $F$ that cover $F_1$.
{
   <For each filter in $F$ a counter is initialized to zero.>
   for <each $A_i$ contained in $F_1$> {
      for <each filter $S$ in $F$ that has a constraint $A_j$ that covers $A_i$> {
         <Increment the counter of $S$>
      };
   };
   $C$:=<all filters in $F$ whose counter is equal to their number of attribute
           filters>
}
```

Fig. 2. Covering algorithm based on counting covering attribute filters

5.1 Matching Algorithm

The naive algorithm separately matches a given notification against all filters to determine the set of matching filters. More advanced algorithms (see Sect. 6 for a detailed overview) try to exploit similarities among the subscriptions. We are investigating several of these algorithms. Here, we present a matching algorithm that incorporates our generic concept that is based on the idea of *predicate counting* [31, 22]. To evaluate *notExists* constraints the names of all attributes that can be contained in a notification of a given type must be known. The algorithm depicted in Fig. 1 determines all filters that match a given notification *n*. Future work will include investigations of other algorithms, e.g. the *key method* and the *tree method* [31, 22].

5.2 Covering Algorithm

The naive algorithm tests each filter against all others sequentially (see Sect. 2.5). Our algorithm is more efficient and derived from the matching algorithm presented above. The algorithm depicted in Fig. 2 determines all filters that cover a given filter F_1 . The covering algorithm is used by the routing algorithm to determine the set of neighbor brokers to which a given filter must be forwarded. For example, an incoming subscription is only forwarded to those neighbors to which no subscription has been forwarded that covers the incoming subscription.

5.3 Merging Algorithm

Our algorithm avoids to test all filters against all others. It counts the number of identical attribute filters to find merging candidates. Fig. 3 shows an algorithm that determines all filters that can be merged with a given filter F_1 .

The further handling of the set of merging candidates depends on the constraints involved. For all constraints discussed in section 3 (e.g. set constraints on simple values) there exists an efficient algorithm which outputs a single merged filter and a set of filters not included in the merger. For other constraints, an optimal algorithm requires exponential time complexity [8]. In this case the use of greedy algorithms or heuristics (e.g. using a predicate proximity graph) seem to be promising.

```
Merging Algorithm
Input:   filter $F_1$, set of filters $F$
Output:  the set $M$ of all filters in $F$ that can be merged with $F_1$.
{
   <For each filter in $F$ a counter is initialized to zero.>
   for <each $A_i$ contained in $F_1$> {
      for <each filter $S$ in $F$ that has a constraint $A_j$ that is indentical
               to $A_i$> {
         <Increment the counter of $S$>
      };
   };
   $M$:=<all filters in $F$ whose counter is one smaller than or equal to their
           number of attribute filters>
}
```

Fig. 3. Merging algorithm based on counting indentical attribute filters

6 Related Work

Siena [6, 7] is a notification service scalable to wide-area networks. It treats the problem of reducing the number of filters to be forwarded by exploiting covering between them. Siena notifications are name value pairs whose values are instances of some predefined primitive types (e.g. integers). Filters are conjunctions of attribute filters whose constraints are either comparisons on the predefined simple values or constraints on textual strings. Algorithms that determine coverage among filters are not presented.

Gryphon developed at the IBM T. J. Watson Research Center uses the content-based matching algorithm presented by Aguilera et al. [1]. This algorithm traverses a parallel search tree where non-leaf nodes correspond to simple tests and edges from non-leaf nodes represent results. Leaf-nodes are associated with matched subscriptions. In [2] a multicast routing algorithm is presented that executes the matching algorithm at each broker. Every broker has a copy of the complete parallel search tree and annotates it to determine the set of neighbor brokers to which a notification has to be forwarded. The algorithm presented is limited to equality tests. In another paper [21] various multicast notification dissemination strategies are compared.

Yan and Garcia-Molina [31] describe the predicate counting, key, and tree algorithms in the context of matching text documents against keyword-based profiles. In this paper they also present performance results obtained from simulation.

In Fabre et al. [11] and Pereira et al. [22] matching algorithms are presented which exploit similarities among predicates. In a first step the satisfied predicates are computed and after that the number of predicates satisfied by a subscription are counted using an association table. Two variants of this algorithm are described which incorporate special treatment of equality tests and of constraints having only inequality tests.

A predicate matching algorithm for database rule systems is presented by Hanson et al. [15] that indexes the most selective predicate that is determined by the query optimizer. They use a special indexing data structure called interval binary search tree to support the efficient evaluation of interval tests.

Gough and Smith [14] present a matching algorithm that is based on automata theory. They show how a set of conjunctions of predicates, each dependent on exactly one attribute, can be transformed to a deterministic finite state automaton. In the paper different types of test predicates are considered and complexity results are obtained. Their algorithm is very efficient, but its worse case space complexity is exponential. The proposed solution is also not suited for dynamic environments as the automaton has to be newly constructed from scratch if subscriptions change.

In [17, 27] indexing strategies for continual queries based on trigger patterns are presented. In particular, a strategy which uses an index on the most selective predicate is described. More complex indexing strategies exploit similarities among trigger patterns to reduce the processing costs. They restrict optimizations to constraints which place a constraint on a single attribute involving at most one constant.

7 Implementation

In the context of our research project Rebeca [12], we investigate event-based architectures for electronic commerce applications and trading platforms. We have realized a prototype of an event-based middleware that incorporates the content-based filtering concepts presented in this paper.

Moreover, we have implemented a complete stock trading application based on real-time quotes to investigate the system under an observable load. At the moment, we are developing another application dealing with meta-auctions which are a generalization of normal internet auctions. Finally, a simple demo that shows how to realize self-actualizing web-pages with our middleware can be found on our project homepage.

8 Conclusion

In this paper we have presented a generic approach for content-based publish/subscribe systems. In particular, we described how generic constraint classes can be used to support various value types. The mechanisms introduced are separated from the routing and matching algorithms. Therefore, new constraints and types can be added easily. We have demonstrated that this approach is feasible for a set of important constraints and types that cover a large amount of constraints with practical relevance. Moreover, we introduced a mechanism to merge filters and described some important rules that can be applied. We also presented novel matching, covering, and merging algorithms that support our generic approach.

Currently, we are investigating how to extend the concepts of covering and merging to more complex content-based data models including semistructured data and objects. We are planning to carry out a detailed performance analysis of various filtering strategies.

Acknowledgements

I gratefully acknowledge the fruitful discussions with Alejandro Buchmann, Ludger Fiege, Felix Gärtner, and Christoph Liebig.

References

[1] M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. Matching events in a content-based subscription system. In *PODC: 18th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, 1999.

[2] G. Banavar, T. Chandra, B. Mukherjee, J. Nagarajarao, R. E. Strom, and D. C. Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, 1999.

[3] J. Bates, J. Bacon, K. Moody, and M. Spiteri. Using events for the scalable federation of heterogeneous components. In *Eighth ACM SIGOPS European Workshop: Support for Composing Distributed Applications*, Sintra, Portugal, September 1998.

[4] C. Bornhövd, M.A. Cilia, C. Liebig, and A.P. Buchmann. An infrastructure for meta-auctions. In *Second International Workshop on Advance Issues of E-Commerce and Web-based Information Systems (WECWIS'00)*, San Jose, California, June 2000.

[5] N. Carriero and D. Gelernter. Linda in context. *CACM*, 32(4):444–458, Apr. 1989.

[6] A. Carzaniga, D. Rosenblum, and A. Wolf. Content-based addressing and routing: A general model and its application. Technical Report CU-CS-902-00, Department of Computer Science, University of Colorado, USA, 2000.

[7] A. Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano, Milano, Italy, December 1998.

[8] A. Crespo, O. Buyukkokten, and H. Garcia-Molina. Efficient query subscription processing in a multicast environment. In *Proceedings of the 16th International Conference on Data Engineering (ICDE)*, page 83, 2000.

[9] G. Cugola, E. Di Nitto, and A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In *Proceedings of the 1998 International Conference on Software Engineering*, pages 261–270. IEEE Computer Society Press / ACM Press, 1998.

[10] P. Eugster, R. Guerraoui, and J. Sventek. Type-based publish/subscribe. Technical Report DSC ID:200029, EPFL Lausanne, 2000.

[11] F. Fabret, F. Llirbat, J. Pereira, and D. Shasha. Efficient matching for content-based publish/subscribe systems. Technical report, INRIA, 2000.

[12] L. Fiege and G. Mühl. Rebeca Event-Based Electronic Commerce Architecture, 2000. http://www.gkec.informatik.tu-darmstadt.de/rebeca.

[13] M, J. Franklin and S. B. Zdonik. "Data In Your Face": Push Technology in Perspective. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 516–519. ACM Press, 1998.

[14] K. Gough and G. Smith. Efficient recognition of events in distributed systems. In *Proceedings of 18th Australasian Computer Science Conference (ACSC)*, February 1995.

[15] E. N. Hanson, M. Chaabouni, C.-H. Kim, and Y.-W. Wang. A predicate matching algorithm for database rule systems. In *19 ACM SIGMOD Conf. on the Management of Data, Atlantic City*, pages 271–280, May 1990.

[16] L. Liu, C. Pu, and W. Tang. Continual queries for internet-scale event-driven information delivery. *IEEE Knowledge and Data Engineering*, 11(4):610–628, August 1999.

[17] L. Liu, C. Pu, W. Tang, and W. Han. Conquer: A continual query system for update monitoring in the wwww. *International journal of Computer Systems, Science and Engineering, Special issue on Web semantics*, 1999.

[18] OMG. CORBA event service specification. OMG Document formal/94-01-01, 1994.

[19] OMG. Corba notification service. OMG Document telecom/99-07-01, 1999.

[20] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The information bus—an architecture for extensible distributed systems. In Barbara Liskov, editor, *Proc. of the 14th Symposium on Operating Systems Principles*, pages 58–68, USA, December, 1993. ACM Press.

[21] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman. Exploiting ip multicast in content-based publish-subscribe systems. In J. Sventek and G. Coulson, editors, *Middleware 2000*, volume 1795 of *LNCS*, pages 185–207. Springer, 2000.

[22] J. Pereira, F. Fabret, F. Llirbat, and D. Shasha. Efficient matching for web-based publish/subscribe systems. In Opher Etzion and Peter Scheuermann, editors, *Proc. of the Int. Conf. on Cooperative Information Systems (CoopIS)*, volume 1901 of *LNCS*. Springer, 2000.

[23] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with Elvin4. In *Proceedings AUUG2K*, Canberra, Australia, June 2000.

[24] N. Skarmeas and K. Clark. Content-based routing as the basis for intra-agent communication. In *Proc. of the 5th Intern. Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555 of *LNAI*, pp. 345–362, 1999. Springer.

[25] Sun. Distriubted event specification, 1998.

[26] Sun. Java message service specification 1.0.2, 1999.

[27] W. Tang. *Scalable Trigger Processing and Change Notification in the Continual Query System.* Oregon Graduate Institute, 1999.

[28] TIBCO, Inc. Tib/Rendezvous. White Paper, 1996. http://www.rv.tibco.com/.

[29] J. Vitek, N. Horspool, A. Krall. Efficient type inclusion tests. *ACM SIGPLAN Notices*, vol. 32, pp. 142–157, Oct. 1997.

[30] M. Wray and R. Hawkes. Distributed virtual environments and VRML: An event-based architecture. In *Proc. of the 7th Intern. WWW Conference, Brisbane, Australia*, 1998.

[31] T. W. Yan and H. Garcia-Molina. Index structures for selective dissemination of information under the Boolean model. *ACM Trans. on Database Systems*, 19(2):332–334, 1994.

Supporting Heterogeneous Users in Collaborative Virtual Environments Using AOP[⋆]

Monica Pinto, Mercedes Amor, Lidia Fuentes, and Jose M. Troya

Dpto. de Lenguajes y Ciencias de la Computación, Universidad de Málaga
Campus de Teatinos, s/n. cp. 29071 Málaga (SPAIN)
{pinto,pinilla,lff,troya}@lcc.uma.es

Abstract. Nowadays, the interest in collaborative virtual environments has increased considerably, probably due to the current technological advances specially on Internet computing. Our main goal is to model collaborative virtual environments combining component-based and aspect-based software technologies. In this paper we are going to highlight the innovative design of a middleware layer that is able to bind dynamically different configurations of the same environment according to the users preferences.

1 Introduction

The continuous evolution of data networks infrastructures is making distributed projects more viable [1]. The advances in communication technology and the impact of distributed multimedia systems are making computer support to evolve from individual to group work, enhancing the development of computer supported collaborative work (CSCW) applications. In this paper we are going to focus on collaborative virtual environments (CVE), a particular sort of CSCW systems.

The main goal in the development of large-scale distributed systems is to achieve a high degree of *configurability*, *extensibility*, *scalability* and *adaptability*, in order to cope with the new requirements and changes these systems are continuously affected by. In particular CVEs have to deal with the coordination of many users in the same session, each one setting different preferences related to the global environment. Consequently, CVE applications must be able to adapt the system, both statically and dynamically, to different *user preferences*.

A CVE tool must provide collaboration zones where users are allowed to cooperate with each other by using different collaborative tools. But, the success of a collaborative tool highly depends on how good is implemented the *awareness* issue, that is, the quality and amount of information that helps a user to perceive the organization of the environment. Then, the main goal in the development of these systems is to construct a *secure*, *persistent* and *integrated* shared environment that provides to the geographically dispersed users the awareness

[⋆] This research was funded in part by the CICYT under grant TIC99-1083-C02-01, and also by the telecommunication organization "*Fundación Retevisión*"

© Springer-Verlag Berlin Heidelberg 2001

they need to communicate and collaborate as if they were co-located in a real work place [2]. Most of the research in the development of CVEs in the last years [3] [4] [5] already cope with the *awareness* concern as well as other important issues that arise in CVEs, like *persistence*, *authentication*, *collaboration* and *user preferences*. However, these approaches do not consider the modeling of these features outside the entities that model the basic functionality of the application domain, resulting in more difficult implementation and code reuse.

Most of the current approaches in CVEs propose implementations of *virtual offices* over the Web, that are *virtual work environments* where the users collaborate at different times and different places. However, these approaches pay more attention to the graphical visualization of the environment than to the software architecture of the system. Our goal is a bit more challenging as we want to provide the appropriate software architecture to model any real-work environment in which people collaborate to carry out a task, being *virtual offices* just an example, and focusing on the definition of a frame architecture flexible enough to deal with evolution. In order to cope with the complexity of developing CVE frame architectures it is very important to choose the appropriate software technologies.

Recently, the emergence of CBSE (Component-Based Software Engineering) [6] imposed a new way of developing applications trying to reuse and compose software components, specially those which were purchased on the COTS (Components Off-The-Shelf) market. Applying this paradigm and trying to avoid the development of a new environment from scratch, Component Application Frameworks (CAF) provide the skeleton of an application based on components.

In addition, most of the characteristics of CVEs mentioned above are present in more than one component in the system. This causes the code of these properties to be intermingled among them and also with the code of components' basic functionality. An additional non-desired consequence is that the code of the same property is spread across different components. COAFs fail dealing with these kind of issues, because it is tough to split the system in decoupled components that are expected to be reused in different contexts. Suppose we want to implement a component that models a *room* and we have two possible *levels of persistence* - object persistence where the room component is serialized and stored to be restored later, and action persistence where only the initial state and the actions executed over the component are stored, and three different *graphical representations* - 2D, 3D and virtual reality. If we implement these issues inside the *room* component we will need 2*3 different implementations. In addition, if we want to add a new issue, for example *awareness*, we will have to extend the six previous implementations for each different implementation of *awareness*, creating interdependences that make difficult their extension and modification.

We have found that the solution to the above problem is the use of a software technology that allows us to separate components and the issues that cut across them in two different entities. This technology is *Aspect-Oriented programming* (AOP) [7] [8] [9] and the two entities are *components* and *aspects*. The separation of components' main functionality and aspects present in more that one component, results in more *reusable*, *configurable* and *extensible* systems. This fact makes easier the instantiation of the environment based on the *user preferences*, the main topic of our approach.

In this paper we try to identify some of the most characteristics issues in CVEs and how we can model these issues to make the environment as configurable and extensible as possible. In this approach we combine component-based and aspect-based software technologies in the development of CVE systems that can be automatically instantiated and configured at runtime based on the *user preferences*. In the next sections we will present the main characteristics of CVEs and our proposal to construct these systems. The most significant part that supports this approach is a middleware layer that performs the dynamic composition between components and aspects at runtime. In a previous work [10] we modeled successfully the *coordination* aspect in a framework for the development of distributed multimedia applications called MultiTEL [1] . The principal novelty of the work present here is that we model other aspects in addition to the *coordination* one, like *awareness*, *persistence*, *authentication* and *multiples views*. We are going to highlight the innovative design of the middleware layer that is able to bind dynamically different configurations of the same environment depending on the *users preferences*.

2 CVE Systems

The term CSCW embraces a great variety of applications. Any asynchronous or synchronous application that makes possible the collaboration among different users to accomplish a common task, falls in the range of CSCW systems. Some examples are *electronic mail*, *chats*, *whiteboards*, *shared editors* and *videoconference tools*. CVEs can be seen as a step forward in the development of collaborative systems where the applications mentioned before are integrated together within a *shared environment*. The environment models the same artefacts that are commonly used in the physical places where users work daily - rooms, file systems, documents, individual tools, collaborative tools, workers and clients among others.

Everyone agrees that the development of any distributed and open system is not a straightforward task. In the construction of CVEs we have to take into account firstly *coordination*, *communication*, *replication*, *security* and *synchronization* concerns, that are typical features of distributed systems. Secondly, more specific issues of CVEs must be modeled with care, trying to ease the construction of *configurable*, *extensible* and *adaptable* systems. These issues are described below:

1. *Awareness*: The provision of other members' awareness is essential in a collaborative environment. It is very important to select carefully the information transmitted, because the system must provide awareness, but with-

[1] MultiTEL Web Site: http://www.lcc.uma.es/~lff/MultiTEL/

out invading the privacy of the sender or creating disturbance for the receiver [11]. In a CVE there must be a minimum amount of awareness information that has to be transferred between users to assure the coherency of the environment. However, each user might be able to configure the *awareness level* that he or she wants to provide to and receive from others members in the environment, both statically and dynamically.

2. *Persistence*: Persistence is also an important issue in a CVE. The environment itself must be persistent since it must exist even if no user is connected to it. Other data that must be persistent is the configuration of the environment per user or the users' actions, for instance, modifications in the user profile, or creation and updates of documents. In order to increase the flexibility of the system, the level of persistence for each component in the environment should also be configurable.
3. *Multiple views*: We want to give the option of having different views of the environment depending on the *users preferences* and resource availability. This means that the user may change the appearance of a component at runtime.
4. *Collaboration*: Collaboration is the main issue in a CVE. The environment must provide all the needed resources to allow the users to work in teams like in a real work place. Two basic resources are the appropriate level of *awareness* to know where are the other members and when they are available to collaborate and the *collaborative tools* they need to work together.
5. *Authentication*: The resources in the CVE have owners and indeed some access control. Some objects are owned by the environment, being the system which establishes access permissions. Other objects are owned by specific users who will be able to change the access permissions at runtime.

One of our main goals is that some of the above issues, like *awareness*, *persistence* or *multiple views* were established by the *user preferences*. When a user enters the first time in the environment, he or she can configure the environment visualization and the level of awareness and persistence that he or she prefers among many other decisions. These preferences will be persistent, so if the user connects later from the same computer or from any other one, his or her configuration will be recovered. This means that the CVE must be prepared to deal with different configurations for each user site and with the modification of them both statically and dynamically.

In the next sections we present an AOP approach to achieve such a high configurable environment. A justification of these can be found in [2].

3 Benefits of Aspect-Oriented Programming

Nowadays, one of the wide used software technology for the development of complex systems is the component-oriented paradigm (COP). During the design phase, the developer must examine the application domain to analyze the main functionalities and encapsulate them in *components*. The components must be as decoupled as possible to increase their *reusability*, *extensibility* and *adaptability*.

By the other way, there are some issues that are normally considered during the implementation of the components but they are not present in the design phase. Some of these issues are intrinsic to distributed systems and others are specific of the application domain, as we showed in the previous section. In COP these issues are implemented across different components decreasing their reusability and increasing the code replication. System developers know the importance of, for instance, *synchronization*, *persistence*, or *security* issues but normally they are just coded as part of the components.

Aspect-Oriented Programming tries to solve this problem with the separation among basic functionality (components) and properties that interact with this functionality (aspects). The main contribution of AOP is not the aspects themselves, but the modeling of these aspects as separate entities. Let us go now to see an example, to show the advantages of using AOP in conjunction with COP instead of just using COP.

3.1 Example: Access Control Using AOP

Suppose we have a *virtual office* with different *rooms* or *places* where the users can move around to collaborate accessing to different *documents* and *tools*. The *environment* itself, the *rooms*, the *documents* and the *tools* are different resources in the virtual office represented as different components.

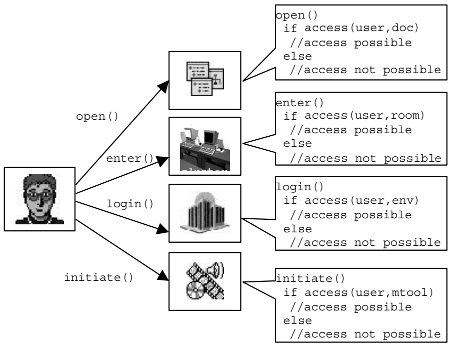
When a user opens a connection to the environment he or she will have different permissions depending on the user profile. These permissions will affect the resources the user can access and the operations he or she can do over them. For instance, an administrator user will not have the same access control than a guest user. Now, we are going to present the differences between using and not using AOP in the design of the *access control policy* for a CVE.

In Fig. 1, as a user *opens* a document, *enters* in a room, *logins* into the environment or *initiates* a multimedia tool, each component checks if the user has access to it. This means that all these components have the same code replicated, because *access control* cut across most of the components in the system.

The replication problem presented here can be solved if we model the *control access* as an *aspect*. As we can see in Fig. 2, the *user*, *document*, *room*, *environment* or *tool* components do not have to take into account nothing about control access. When the user wants to interact with any of these components, the *authentication aspect* intercepts the request and checks if the user has permission to access to the target component. If the user has not the right permissions, the aspect does not send the request to the target component.

4 Our CVEs Development Proposal

Our proposal is the construction of a system for the development of CVEs where the components and aspects are composed dynamically at runtime through a *middleware layer* that offer several services, *VESite* among them. Figure 3 shows the architecture of a client site, where the aspects and component objects reside



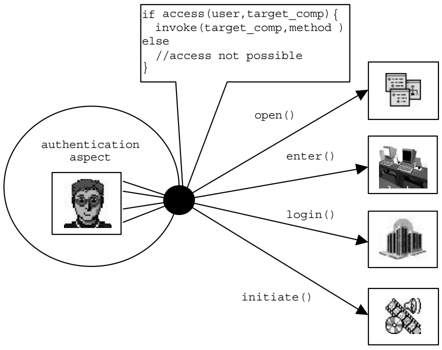
```
open()
 if access(user,doc)
  //access possible
 else
  //access not possible
```

```
enter()
 if access(user,room)
  //access possible
 else
  //access not possible
```

```
login()
 if access(user,env)
  //access possible
 else
  //access not possible
```

```
initiate()
 if access(user,mtool)
  //access possible
 else
  //access not possible
```

Fig. 1. Access control with a traditional approach



```
if access(user,target_comp){
  invoke(target_comp,method )
else
  //access not possible
}
```

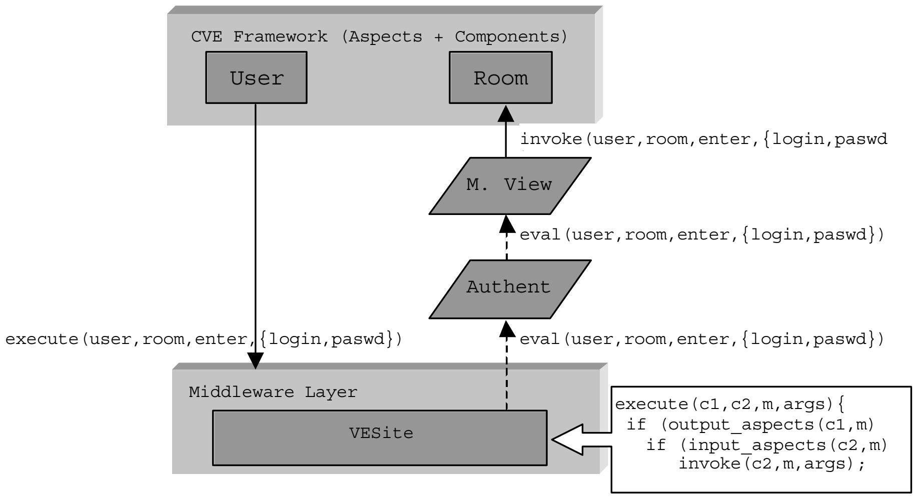
Fig. 2. Access control using AOP

in the first level and are composed at runtime by the *VESite* of the middleware layer. As showed in Fig. 3, when a component (*User*) wants to send a message (*login*) to another component (*Room*) it sends it through the middleware layer using the *execute* method in the VESite component. This component retrieves the information about the aspects that must be evaluated before the message reaches the target component and evaluates them transparently to the source and target components.

In this paper we focus in the *VESite* component, the most important part of the middleware layer, that stores the information required to make possible the dynamic composition between components and aspects. We are going to show how it is possible to manage different configurations depending on the *users preferences*.

4.1 The Virtual Environment Site

Previously, we mentioned that the composition between components and aspects in our system is performed at runtime. This implies several points:


CVE Framework (Aspects + Components)
User
Room
invoke(user,room,enter,{login,paswd
M. View
eval(user,room,enter,{login,paswd})
Authent
eval(user,room,enter,{login,paswd})
execute(user,room,enter,{login,paswd})
Middleware Layer
VESite

```
execute(c1,c2,m,args){
 if (output_aspects(c1,m)
   if (input_aspects(c2,m)
      invoke(c2,m,args);
```

Fig. 3. CVE development system

– Both components and aspects are first order entities that will exist at run-time.
– Both components and aspects are represented by an unique identifier and they do not have direct references to other components or aspects.
– Components have no knowledge about the aspects they are affected by.
– The number and type of aspects that are applicable to a component can change dynamically.
– The aspects that are applied to a sort of component can be different for each user in the same environment, due to the *user preferences*.

In order to accomplish these points we define a special component (*VESite*), that represents each user in the CVE, and stores all the information needed to compose components and aspects dynamically, that is, it determines the system's configuration.

The *VESite* is divided in two objects as it is shown in Fig. 4, the *EnvironmentSite* and the *UserSite*. The *EnvironmentSite* object stores the list of all the *components* and *aspects* that can be instantiated in the environment and the *architectural constraints* for all the users in the CVE - which aspects can be applied to each component and in which order they are applied. The *UserSite* object stores the specific configuration for one user of each component and aspect and the *CVE application context* with the references to the components and aspects instantiated for that. So, we put in a single class all the information related to a specific user, according to his or her preferences, as a particularization of the common behavior.

The first time a user opens a connection to the CVE application, the environmental information is loaded from the *EnvironmentSite* object. Then, the CVE application can be configured by each user according to his or her preferences.

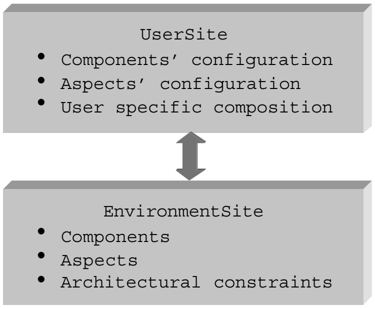

UserSite
Components' configuration
Aspects' configuration
User specific composition
EnvironmentSite
Components
Aspects
Architectural constraints

Fig. 4. The virtual environment site

Each decision taken by the user takes effect immediately and also is stored in the *UserSite* object. When the user leaves the environment the *UserSite* is saved, and it is restored the next time the user connects to the environment.

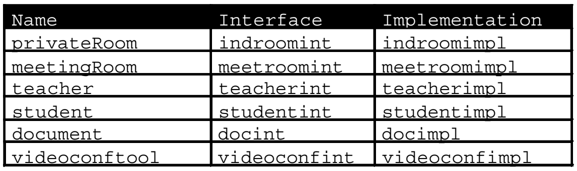
As we mentioned before, some of the features that can be configured are the *awareness* level, the *persistence* level and the *multiple view* issue. If these features are modeled by aspects this implies that different users will be using different configurations of the same aspects and even different implementations of them. For instance, a user can configure the environment as a 3D virtual environment while other user with fewer resources will configure it with a 2D representation. Both users can collaborate and work together because the components' behavior are exactly the same, only their graphical representation is different.

The same can be applied to the components. For instance, a user can make their actions persistent using a component that stores the information in a database while other user may choose to store this information simply in a file. Since the components interfaces are the same, both can play the same role inside the architecture of the system.

The EnvironmentSite Component. This component maintains the architecture of the environment defined in terms of the components and aspects that can be instantiated, and the architectural constraints to compose them. The architectural constraints comprise the restrictions about the composition rules between components and aspects, and the order in which the aspects can be applied. We have defined in [12] a script language that a designer can use to describe the architecture of a CVE application. Each description is compiled and verified, checking the information consistency. All this information is then placed inside this object, implemented as private classes. Now, we are going to describe this architectural information through some tables, with an example.

We are going to show a partial architectural description (components & aspects) of a virtual environment that models a university department.

1. *Components.* Figure 5 shows the list of components that can be part of the environment, for instance, different kind of *rooms* components where



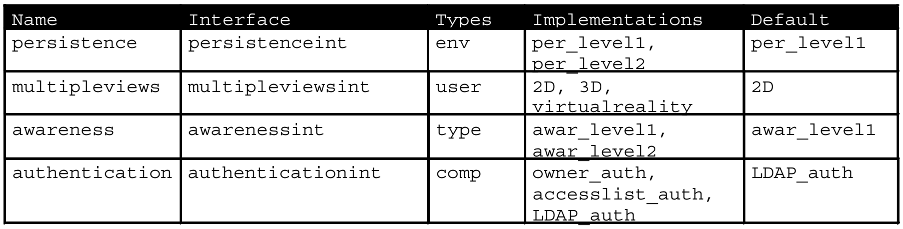
Name	Interface	Implementation
privateRoom	indroomint	indroomimpl
meetingRoom	meetroomint	meetroomimpl
teacher	teacherint	teacherimpl
student	studentint	studentimpl
document	docint	docimpl
videoconftool	videoconfint	videoconfimpl

Fig. 5. Component definition

users meet them and collaborate, different users components (e.g. teacher, student), a *document* component and a *videoconference tool*. We identify each component by a *name* that will be an unique string identifier inside the framework, its *interface* and its *implementation* class.

2. *Aspects*. In Fig. 6 it is represented the list of aspects that can be instantiated in the CVE with a set of attributes. Likewise the components, the aspects are defined by a unique *name*, the *interface* that implements, the *type* of the aspect, a set of *implementations* and the aspect that it is going to be used by *default*.
   In our architecture an aspect is assigned to one of the following types, according to the number of instances of that aspect that will be able to create inside the environment:
   - *Environment-oriented* (env): Only one instance of the aspect in the CVE.
   - *User-oriented* (user): Only one instance of the aspect for each *VESite*.
   - *Type-oriented* (type): One instance of the aspect for each type of component.
   - *Component-oriented* (comp): One instance of the aspect for each component instance.

   An example of an *environment-oriented* aspect is the *persistence* aspect if the organizer of the CVE decides to use the same data store for all the information that needs to be persistent and for all users in the CVE. We classify the *multiple views* aspect as *user-oriented* because there will be only one instance of this aspect for all the components associated to a single user, due to all his or her components should have the same graphical representation. The *awareness* has been defined as a *type-oriented* aspect since depending on

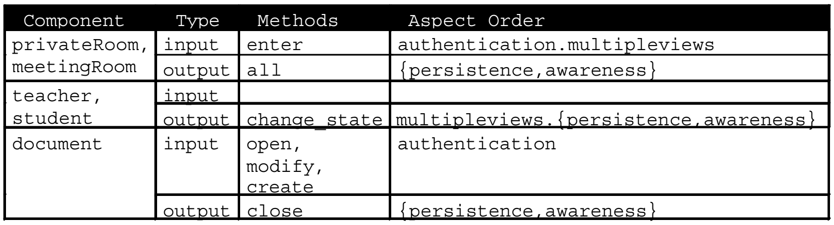


Name	Interface	Types	Implementations	Default
persistence	persistenceint	env	per_level1, per_level2	per_level1
multipleviews	multipleviewsint	user	2D, 3D, virtualreality	2D
awareness	awarenessint	type	awar_level1, awar_level2	awar_level1
authentication	authenticationint	comp	owner_auth, accesslist_auth, LDAP_auth	LDAP_auth

Fig. 6. Aspect definition

the component's type the awareness information will vary. For instance, the *user* awareness is his or her location and the job he or she is doing at each moment, while the *document* awareness is its location, the last modification date and the person who last worked on it. However, the *authentication* aspect is a *component-oriented* aspect because each component might have its own authentication method depending on the *user preferences* or the role he or she plays inside the environment (i.e. students or teachers).
We put different implementations for the same aspect because each user could prefer different levels of, for instance, *persistence* or *awareness*. In the *EnvironmentSite* we provide information about all the available implementations. Afterwards, the user selects through configuration options the one he or she desires. The *default* denotes the minimum level for one aspect that should be present inside the environment. For instance, the user can select between different levels of *awareness*, but the environment developers must assure that at least the location of each user in the environment must be always available. In order to guarantee it, an architectural restriction must be the use of the awareness implementation *awar_level1*. If the *default* field is empty it means that the use of that aspect is not compulsory and each user can decide whether to use it or not.

3. *Composition rules* between components and aspects. In Fig. 7 it is showed which aspects can be applied to each component. An aspect can be an *entry* or an *exit aspect* (input and output in Fig. 7). An *entry aspect* is evaluated just before the execution of a method in a component and an *exit aspect* is evaluated right after the execution of the method. The *composition table* indicates, for each component, which aspects can be applied to it, both the *entry* and *exit aspects* and also to which methods of the component they are applied . The use of the *all* wild-card is possible to indicate all the components or all the methods.
4. *Order* in which the aspects are applied. The aspects invocation order must be specified both for *entry* and *exit* aspects because it may vary. In Fig. 7 we use the following syntax in order to express the order:
   – The aspects are expressed through a list of items separated by the sequential symbol ".".
   – The order of application is the same that the order of appearance in the list.



Component	Type	Methods	Aspect Order
privateRoom, meetingRoom	input	enter	authentication.multipleviews
	output	all	{persistence,awareness}
teacher, student	input		
	output	change_state	multipleviews.{persistence,awareness}
document	input	open, modify, create	authentication
	output	close	{persistence,awareness}

Fig. 7. Component-Aspect composition

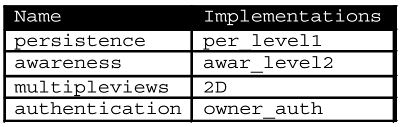
– If several aspect can be applied in any order we put them between the symbols "{}".

For instance as it is seen in Fig. 7, for the components *privateRoom* and *meetingRoom*, the *entry* aspects are applied to *all* methods. The order of application is in first place the *authentication* aspect to check if the component that is trying to interact with the *room* has access. The typical components that interact with the *privateRoom* or the *meetingRoom* components are going to be the components modeling the users. Then, the second aspect applied will be the *multiple views* aspect, followed by the *persistence* and *awareness* aspects that can be applied in any order.

The UserSite Component. In the *EnvironmentSite* component, the developer specifies the global architectural restrictions of an application. However, the user must make some decisions to tailor the system configuration. For instance, the user must decide the level of awareness or persistence among all the implementations offered by *EnvironmentSite* as shows Fig. 8. Otherwise, the *EnvironmentSite* set the *default* implementation for these aspects.

The decisions taken by each user can be different, so we define a second level, the *UserSite* to store all the user settings about the environment. The *UserSite* component contains:

1. *CVE user preferences*: There are different types of *user preferences*. Some of them affect the behavior of the user in the environment, for instance, the level of *persistence*, *level of awareness*, *graphical representation* (2D, 3D, virtual reality) or the *authentication* mode (Fig. 8). Actually, the user does not choose between implementation classes, but among sets of preferences displayed in a menu that correspond to a specific class. The *UserSite* contains this personal information for each user.
   Other properties with regard to the visualization of the components in the environment, for each user, but it does not change its behavior. For instance, the windows color, size and distribution in the screen. Once the user establishes these properties, is the *multiple views* aspect which is in charge of obtaining them to configure each component as desired by the user.
2. *Components & Aspects configuration values*: The *UserSite* stores all the information needed by the components and aspects. For instance, in Fig. 8 we can see that the control access is decided by the user (owner_auth). As a consequence of this the *authentication* aspect needs to know who is the *owner* and this information is stored in the *UserSite* component.



Name	Implementations
persistence	per_level1
awareness	awar_level2
multipleviews	2D
authentication	owner_auth

Fig. 8. CVE preferences for a concrete user

3. *CVE application context*: The *UserSite* component must store the actual *context* of the user it represents. The *CVE application context* consists on the references of all the components and aspects instantiated in each moment for that user. This context will change dynamically as the components and aspect instances are created or destroyed during the execution of the application.

The concrete values of *user preferences* must remain in some place to be used next time. We need a storage service, for instance, a database or a LDAP Directory Service, to keep the properties of all the resources in the CVE. The *UserSite* component downloads the *user preferences* in the initialization method, and afterwards they will be requested by aspects or components as soon as they need it.

5 Conclusions and Future Work

In this paper we have presented the main characteristics of CVEs and the suitability of using aspect-oriented programming in conjunction with component-oriented programming in the development of this type of systems.

The main issues that are present in CVEs - *awareness*, *persistence*, *authentication*, and *multiples views*, are modeled in other approaches as part of the basic functionality of the components that model the main entities of the environment. Our principal contribution is the separation of these concerns in *aspects* and the composition between components and aspects at runtime. We have shown that AOP is fine not only for *synchronization* and *coordination* aspects, but for other sort of issues.

In addition, the middleware layer, that allows the dynamic composition between components and aspects, provides a powerful mechanism to bind different configurations of the same environment depending on the *user preferences*.

Our future goals are to complete the definition of the middleware layer, the definition of the architecture of the CVE framework and the implementation of a working prototype, concretely we are developing a virtual office as part of a funded research project.

References

1. E. Ly.: Distributed Java Applets for Project Management on the Web. 4th International Workshop on Component-Oriented Programming WCOP'99 in conjunction with the European Conference on Object-Oriented Programming ECOOP'99. June 1999. 226
2. M. Pinto, M. Amor, L. Fuentes, J. M. Troya.: Collaborative Virtual Environment Development: An Aspect-Oriented Approach. Proceedings of DDMA Workshop. Phoenix, Arizona, April 2001. 227, 229
3. M. Roseman and S. Greenberg.: Teamrooms: Network Places for Collaboration. Proceedings of ACM CSCW. 1996. 227

4. H. Shinkuro, T. Tomioka, T. Ohsawa, K. Okada, and Y. Matsushita.: A Virtual Office Environment based on a Shared Room realizing Awareness Space and transmitting Awareness Information. Proceedings of the 10th annual ACM symposium on User Interface Software and Technology. 1997. 227
5. M. Sohlenkamp and G. Ghwelos.: Integrating Communication, Cooperation and Awareness: The DIVA Virtual Office Environment. Proceedings of ACM CSCW. 1994. 227
6. A. W. Brown, K. C. Wallnau.: The Current State of CBSE. IEEE Software. September/October, 1998. 227
7. C. A. Constantinides, A. Bader, T. H. Elrad, M. Fayad, and P. Netinant.: Designing an Aspect-Oriented Framework in an Object-Oriented Environment. ACM Computing Surveys. March 2000. 227
8. G. Kiczales et al.: Aspect-Oriented Programming. Proceedings of ECOOP'97. LNCS 1241. Springer-Verlag. 227
9. C. Lopes, E. Hilsdale, J. Hugunin, M. Kersten, and G. Kiczales.: Illustrations of crosscutting. ECOOP 2000 Workshop on Aspects & Dimensions of Concerns. June 11-12 2000. 227
10. L. Fuentes and J. M. Troya.: Coordinating Distributed Components on the Web: an Integrated Development Environment. Software-Practice and Experience. 31, 2001. 228
11. S. E. Hudson and I. Smith.: Techniques for Addressing Fundamental Privacy and Disruption Tradeoffs in Awareness Support Systems. Proceedings of the ACM, conference on CSCWI. 1996. 229
12. M. Pinto, M. Amor, L. Fuentes, J. M. Troya.: Towards an Aspect-Oriented Framework in the Design of Virtual Environments. Submitted to Third IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems. September, 2001. 233

A Process Service Model for Dynamic Enterprise Process Interconnection

Karim Baïna, Khalid Benali, and Claude Godart

LORIA - INRIA - CNRS - UMR 7503
BP 239, F-54506 Vandœuvre-lès-Nancy Cedex, France
{baina,benali,godart}@loria.fr

Abstract. Due to business process automation development, process interconnection becomes an important matter. Actually, process interconnection mechanisms are indispensable to co-ordinate business processes within and beyond organisation boundaries, to strength awareness inside virtual enterprises, to facilitate multinational e-transactions, etc. Therefore, thinking and proposing mechanisms to insure interconnection between organisational business processes is becoming a hot topic of research. Actually, existing work co-ordination systems (workflows, shared agenda, co-operative authoring tools, revision and configuration control systems, etc.) have been mainly developed to suit the intern needs of enterprises. Thus most of these systems are not adapted to inter-enterprise co-operation. As we are interested in workflow process co-operation, we aim, through this paper, to provide a model supporting dynamic cross organisational process interconnection. This paper introduces our process model, then our process service model, and finally our process service interconnection model.

1 Introduction

Our purpose is to provide a framework to support dynamic cross organisational process interconnection. We aim to conceive a system to support co-operative process interconnection with neither previously determined communication primitives, nor previously scheduled points of rendezvous. This means that an organisation, aiming to interconnect its work process with another organisation work process (e.g. for outsourcing of a piece of specific software development, for an online command of a service, for data exchange rendezvous in a virtual enterprise, etc.), has to co-decide an interconnection paradigm at runtime. To allow such a kind of interconnection between processes, negotiation mechanisms are, among other mechanisms, necessary to support common decisions between processes (services to exchange, rendezvous, treaties, contracts, needed results,...). Following our service negotiation model [1], this paper presents a process service interconnection model to support dynamic enterprise process interconnection. This paper is structured as follows: section 2 presents the problematic and situates our approach, section 3 formalises our process service interconnection model, and section 4 concludes and gives some perspectives.

© Springer-Verlag Berlin Heidelberg 2001

2 Process Interconnection

Due to business process automation development, process interconnection becomes an important matter. If a wide spectrum of tools for work co-ordination exists (workflows, shared agenda, co-operative authoring tools, revision and configuration control systems, etc.), they have been developed to suit the intern needs of enterprises, and thus, are not adapted to inter-enterprise co-operation. Concerning workflow management system (WFMS) interconnection, some existing interconnection solutions are mostly proprietary (i.e. based on : specific business process definition languages (BPDLs), particular WFMS platforms, private data exchange formats,. . .). To improve generic process interconnection support within existing WFMSs, new interconnection models are being developed. These models deal either with awareness, data and control flow formalisation between two interleaving processes (i.e. dataspace sharing models [2], dataspace replication models [3], message passing mechanism [4,5], event subscription/notification paradigm [6,7,8], remote object invocation [9,10], transfer protocol extension [11], . . .), or with interleaving process life cycle control (i.e. transactional exchange protocols [12,13,14], . . .). They mainly focus on data communication protocols and make abstraction of process co-operation structure and semantics [15]. A process interconnection model aiming to go beyond limitations of current WFMSs and process interconnection frameworks, ought to treat the following compromises:

- **Openness versus Privacy:** For more co-operation transparency and awareness, companies need interconnection points between their processes. However, being directly or indirectly concurrent, they need interconnection mechanisms insuring process privacy aspects (i.e. distinction between shared and private knowledge as well as visible and hidden access points).
- **Scheduled versus Dynamic co-operation:** In the same work conditions, two companies do not co-operate the same way during two similar projects. It turns out that building a model describing all interaction possibilities between two processes is too complex (i.e. combinatory explosion) [15].

Our aim is to develop a service based process interconnection model. Service concept has been defined in many research fields: Object Oriented research [16], Process Modelling research[15,17,18,19,20,21], Distributed System research [22,23], etc. In the field of workflow research, a process service can be seen as a software entity presenting process particularities and outcomes without totally revealing the process structure (i.e. its workflow implementation). A process service shows a functional abstraction of a process (or parts of a process) provided by an organisation. It specifies the amount of work that the organisation promises to carry out with a specific quality of service. It also specifies which parts of a workflow it covers and how the requester could access to them. Service concept has been studied from several point of view: process service execution semantics abstraction [15], sub-workflow process service selection [19], dynamic process service activities configuration [20], process service control flow level abstraction [21], service methods and events wrapping [23], etc. Our aim is to develop a process

service structure as a co-operation design pattern that relevantly supports dynamic process co-operation and interconnection mechanisms. Our process service interconnection model will enable co-operating organisations to structure, classify, and compare process services **(profiling and matching of process services)**, to select dynamically a provided process service among those matching a required process service **(selection and negotiation of process services)**, and finally to keep possible their business processes co-operating through service wrapping **(interconnection of process services)**. For instance, process service may concern either long e-transactions (e.g. outsourcing the development of pieces of software, subscription to full e-learning sessions, etc.), or short e-transactions (e.g. online book commands, enactment of administrative processes, data exchange rendezvous in a virtual enterprise, etc.). To illustrate our approach, let us consider the following example within an e-learning context:
Example 1 Let R (an e-learning enterprise) be a service requester enterprise. Let P_1 (a web agency enterprise), P_2 (a site hosting enterprise), and P_3 (an e-learning content collection enterprise) be three (among other) service provider enterprises. In the first hand, R requires three types of services: a portal development service s_1, a portal hosting service s_3, and an e-learning content collection service s_4. In the second hand, P_1 proposes an internet site development process service s_{11}, P_2 proposes an internet site hosting process service s_{22}, and P_3 proposes an e-learning content construction process service s_{32}. Requested services can be implemented by several provided services. After process service matching and negotiation sessions, R chooses P_1 with its process service s_{11} which matches s_1, P_2 with its process service s_{22} which matches s_3, and P_3 with its process service s_{32} which matches s_4:

3 Our Process Service Interconnection Model

The formalisation of our process service interconnection model is based on process model and process service model.

3.1 Process Model

Our process interconnection model is initially based on the F. Leymann and D. Roller process definition model [24]. If this model can be applied very well for traditional workflows (under one enterprise), it does not consider explicitly process interconnection. Our objective is to enrich this model with new concepts and definitions in order to support some process interconnection aspects. Most of studied interconnection process models focus on process control flow and data flow definition without caring about two important process access points: process instance methods and process instance notification events[1]. For this purpose, we develop the following concepts: definitions 1, 2, and 3 define process, process

[1] To avoid current WFMS heterogeneity, we will consider, in this paper, that WFMSs are compliant with W*f*MC standards, or at least share a common API based on W*f*MC Interface 2 [25].

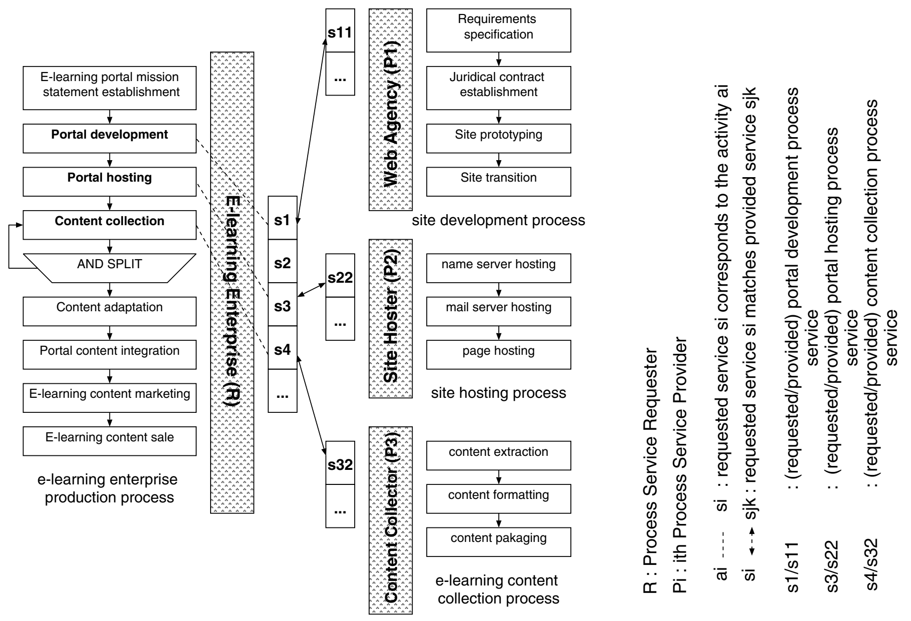

E-learning portal mission statement establishment
Portal development
Portal hosting
Content collection
AND SPLIT
Content adaptation
Portal content integration
E-learning content marketing
E-learning content sale
e-learning enterprise production process
E-learning Enterprise (R)
s1
s2
s3
s4
...
s11
...
s22
...
s32
...
Web Agency (P1)
Site Hoster (P2)
Content Collector (P3)
Requirements specification
Juridical contract establishment
Site prototyping
Site transition
site development process
name server hosting
mail server hosting
page hosting
site hosting process
content extraction
content formatting
content pakaging
e-learning content collection process
R : Process Service Requester
Pi : ith Process Service Provider
ai ---- si : requested service si corresponds to the activity ai
si ◄--► sjk : requested service si matches provided service sjk
s1/s11 : (requested/provided) portal development process service
s3/s22 : (requested/provided) portal hosting process service
s4/s32 : (requested/provided) content collection process service

Fig. 1. An e-learning entreprise process service interconnection example

graph, and process API structures. Definitions 4, 5, and 6 describe process methods, process instance reading methods, and process instance events concepts. A process is composed of a process graph and a process API as follows:

Definition 1 (*P*: Process) *We define a process P as the 2-uplets such that* $P = (G(N,\ E),\ ((Methods,\ \le),\ (Events,\ \le)))$*, where* $G(N,\ E)$ *is the process graph as defined in definition 2, and* $((Methods,\ \le),\ (Events,\ \le))$ *is the process API as defined in definition 3.*

The process graph definition describes the process control-flow structure [24]:

Definition 2 ($G(N,\ E)$: Process graph) *A process graph is a pair* $G(N, E)$*, where N is the set of nodes (activities), E is the set of control connectors (oriented arrows) of the graph ;* $(a \in N, b \in N, c \in C) \in E$ *means that there is a control connector weighted by a condition c coming from a to b. C : set of all conditions (business rules of Boolean functions or predicates).*

A process API is the pair gathering process methods and process events:

Definition 3 (Process API) *We note Process API the pair combining process Methods and Events: Process API* $= ((Methods,\ \le),\ (Events,\ \le))$.

Methods set gathers all process instance reading and updating methods:

Definition 4 (*$Methods$*: Process methods) *Let $Methods$ the set of all process API procedures and functions, more formally $Methods$ is the union of both sets Process instance reading methods ($R-methods$) and Process instance updating methods ($U-methods$). $Methods = R-methods \cup U-methods$.*

Process instance reading methods are process primitives permitting to read (without alteration) a process instance (or one of its attributes) during its execution:

Definition 5 (*$R-methods$*: Process instance reading methods) *Let $R-methods$ the set of process instance reading access methods (e.g. fetchProcessInstanceState(..), fetchActivityInstanceState(..), fetchActivityInstanceAttribute(..)).*

Process instance updating methods $U-methods$ are process primitives that permit to update a process instance (or one of its attributes) during its execution. Due to "Openness versus Privacy" compromise (c.f. section 2), we will focus mainly on $R-methods$ rather than $U-methods$ set[2]. Actually, we assume that, after enacting a service, keeping possible to an extern observer to alter directly the execution of an inner process instance is prohibited.

To construct a totally ordered set $(R{-}methods, \leq) = \{m_1, .., m_{|R-methods|}\}$[3], we enrich $R-methods$ set with a total order relation $\leq$ which measures the privacy of a process instance method inside $R-methods$ set such that:

$$\begin{gathered} \leq \;:\; R-methods \times R-methods \rightarrow Boolean \\ (m_1 \leq m_2) \Leftrightarrow (m_1 \text{ is as confidential as } m_2 \vee \\ m_1 \text{ is less confidential than } m_2) \text{ for an extern observer.} \end{gathered}$$

The total order relation $\leq$ will permit to compare every $R-methods$ element and then enable services to be totally compared and selected according to their proposed API methods via a negotiation tactical (see section 3.3). To enable the process service provider to define such a total order relation, one may classify $R-methods$ in k classes of privacy $(C_i)_{i=1,..,k}$ such that:

$$\begin{gathered} 1 \leq k \leq |R-methods| \wedge \\ \forall\, i \in \{1,..,k\},\ C_i \subseteq R-methods \wedge \\ \forall\, m_1,\ m_2 \in C_i,\ m_1 \text{ is as confidential as } m_2 \wedge \\ \forall\, m \in C_i,\ m' \in C_{i' > i}, \text{ we have } m \leq m' \end{gathered}$$

Example 2 To illustrate $R-methods$ set total ordering approach, let a particular $R-methods$ set with $|R-methods| = 12$ and let k classes of privacy, such as $k = |R-methods| = 12$ as follows:

[2] $U-methods$ will not be detailed in this article.
[3] Let W a finite set, we note $|W|$ the cardinality of the set W.

methods m_i	$R - methods$	**Privacy Classes** C_i
m_1	fetchProcessInstanceState(..)	C_1
m_2	fetchActivityInstanceState(..)	C_2
m_3	fetchWorkItem(..)	C_3
m_4	getProcessInstanceAttributeValue(..)	C_4
m_5	fetchWorkItemAttribute(..)	C_5
m_6	getWorkItem(..)	C_6
m_7	fetchActivityInstanceAttribute(..)	C_7
m_8	fetchActivityInstance(..)	C_8
m_9	getActivityInstance(..)	C_9
m_{10}	getWorkItemAttributeValue(..)	C_{10}
m_{11}	getActivityInstanceAttributeValue(..)	C_{11}
m_{12}	fetchProcessInstanceAttribute(..)	C_{12}

In this $(R-methods, \ \leq)$ configuration, we have $(m_1 \ \leq \ m_2)$ because $m_1 \ \in \ C_1$ and $m_2 \ \in \ C_{2\ >\ 1}$. More than that, we have $(m_1 \ \leq \ m_1)$ and $(m_2 \ \leq \ m_2)$ due to reflexivity of $\leq$ order relation.

As introduced before, process API is composed of process instance methods and events. As we have detailed process methods, we will describe process instance events. Process instance events are notification messages that are triggered off from a process instance (or one of its components) during its execution:

Definition 6 (*Events*: Process instance events) *Let Events the set of all process instance API notification events (i.e. event handling execution status of the process instance and of its components (e.g. TerminatedProcessInstanceNotification), or events handling process instance data production status (e.g. AvailableNewDataNotification)).*

To construct a totally ordered set $(Events, \ \leq) = \{e_1, .., e_{|Events|}\}$, we enrich *Events* set with a total order relation $\leq$. This order relation measures the privacy of a process instance event inside *Events* set, more formally:

$$\begin{gathered} \leq \ : Events \ \times \ Events \ \rightarrow \ Boolean \\ (e_1 \ \leq \ e_2) \ \Leftrightarrow \ (e_1 \text{ is as confidential as } e_2 \ \vee \\ e_1 \text{ is less confidential than } e_2) \text{ for an extern observer.} \end{gathered}$$

A similar $R-methods$ classification approach can be used by the process service provider to totally order *Events* set in $(C_i^{'})_{i=1,..,k^{'}}$ privacy classes.
Example 3 To illustrate *Events* set total ordering approach, let a particular *Events* set with $|Events| \ = \ 5$ and let $k^{'}$ classes of privacy, such a $k^{'}$ may be chosen with respect to $1 \ \leq \ k^{'} \ \leq \ |Events|$. For instance, let $k^{'} \ = \ |Events| \ = \ 5$:

events e_i	*Events*	**Privacy Classes** $C_i^{'}$
e_1	TerminatedProcessInstanceNotification	$C_1^{'}$
e_2	StartedProcessInstanceNotification	$C_2^{'}$
e_3	TerminatedActivityInstanceNotification	$C_3^{'}$
e_4	StartedActivityInstanceNotification	$C_4^{'}$
e_5	AvailableNewDataNotification	$C_5^{'}$

In this ($Events$, $\leq$) configuration, we have ($e_1 \leq e_5$) because $e_1 \in C_1^{'}$ and $e_5 \in C_{5\ >\ 1}^{'}$. More than that, we have ($e_1 \leq e_1$) and ($e_5 \leq e_5$) due to reflexivity of $\leq$ order relation.

In order to define our process model, we have firstly defined process graph concept (definition 2), then, we have defined process API concept (definition 3): where definitions 4 and 5 presented process methods and definition 6 described process instance events.

3.2 Process Service Model

In existing WFMS platforms, only two alternatives of process access rights are possible : *black box* (neither process primitive calls, nor process event notifications can be accessed by an extern organisation) or *white box* (complete API can be accessed: primitives and event notifications). If an organisation estimates that white box alternative is a very careless behaviour (regarding privacy, security, etc.), it switches to black box alternative. Safety of black box alternative is obvious, but it is not acceptable regarding process interconnection purpose. The objective of our process service interconnection model is to make possible the specification of a large process access right panel going from black box to white box solution. For this purpose we develop the following concepts: definitions 7, and 8 describe process service profiling and process service API. Definitions 9, 10 and 11 are concerned with visibility contracts which are access rights on process service API. Definition 12 structures Visibility contract set as paths from blak box to white box visibility contract. Finally, definition 13, and 14 detail concepts of process service matching and neighbourhood. We define process service profile as a functional abstraction of a process service: it presents a meaningful semantics of a process service. This enables the classification, the indexing, the comparison, and the research of a process service. This supposes that enterprises, within each business community, agreed about common process service language (e.g. business key concept ontologies, business service taxonomies,. . .) to define and understand process services. Research and normalisation work are still emergent in this promising field [26,27,28,29]. Based on process concept (definition 1) and visibility contract concept (definition 9), process service profiling describes the structure of a process service:

Definition 7 (Process service profiling) *Let objectClasses the set of all object classes, B the ASCII alphabet, and $T = (\mathcal{R} \cup Boolean \cup B^*)$*[4], *and let E the*

[4] B^* (resp. B^+) represents the language of all (resp. non empty) ASCII strings.

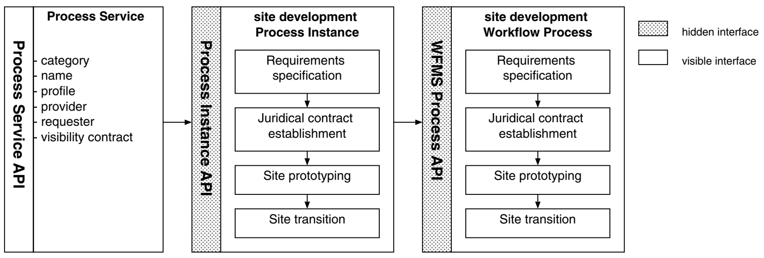


Fig. 2. A Process Service of the Web Agency

co-operating enterprise set. Let S the service set, such that each service $s \in S$ is a 8-uplet ($category \in objectClasses$, $name \in B^+$, $profile \in T^k$, $requester \in E$, $provider \in E$, $process \in P$, $instance$ (*an instance of* $process$), $visibility\ contract \in \overline{V}$)*.*

Process service API is a particular visibility contract (cf. definition 9) assigned to the process service. It is a Process API restriction that is co-decided dynamically between service provider and requester before the process service enactment:

Definition 8 (Process service API) *Let call Process Service API a process service visibility contract $v_i(M_j, EV_k)$ with $v_i \in \overline{V}$. From definition 3, we have Process API* $= ((Methods, \leq), (Events, \leq))$*, therefore, Process service API $\subseteq$ Process API (in the sense of definition 9). This means that a process service API is a restriction of the general Process API.*

Example 4 If we illustrate this service profile schema on $s_1 \in S$ (defined in previous examples), we will obtain for instance after the negotiation of all process service parameters :

s_1 = (category = e_learning_development_portal_service,
name = "e-learning-portal development process service",
profile = (duration=3 (month),
price=100 (Keuro),
dynamic_sites = true,
XML_use = true,
Java_use = true,
JSP_use = true,
flash_use = false),
requester = R (e-learning enterprise),
provider = P_1 (web agency),
process = site_development_process,
instance = site_development_process_instance,
visibility contract = (M_2, EV_3))

Based on Methods Visibility Layers set concept (definition 10) and Events Visibility Layers set concept (definition 11), a visibility contract is a Process

API restriction regarding process methods and events. Predefined black box and white box are particular visibility contracts:

Definition 9 ($\overline{V}$: Visibility Contract set) *Let $\overline{V}$ the visibility contract set be the cartesian product of Methods Visibility Layers set (M) and Events Visibility Layers set (EV). $\overline{V} = M \times EV$, each element $v \in \overline{V}$ represents a restriction (or an access right) on Process API, since it is a combination of a subset of $R-methods$ and a subset of Events.*

Let $\subseteq$ be a partial order relation on $\overline{V}$ such that:

$$\subseteq \; : \overline{V} \times \overline{V} \rightarrow Boolean$$
$$v_i(M_j,\ EV_k) \subseteq vi^{'}(M_{j'},\ EV_{k'}) \Leftrightarrow M_j \subseteq M_{j'} \wedge EV_k \subseteq EV_{k'}$$

Based on $R-methods$ ordered classification, Methods Visibility Layers set structures process instance reading methods in order to keep possible to negotiate a particular $R-methods$ subset (methods layer) for process service interconnection:

Definition 10 (M: Methods Visibility Layers set) *We define Methods Visibility Layers set M as: $M \subset \wp(R-methods)$*[5] *such that $M = \cup_{i=0}^{|R-methods|}\{M_i\}$ where $M_i \subseteq (R-methods, \leq) = \{m_1, .., m_{|R-methods|}\}$, $M_0 = \emptyset$, and $M_{1\leq i\leq |R-methods|} = M_{i-1} \cup \{m_i\}$. Thus, $M = \{\{\}, \{m_1\}, \{m_1, m_2\}, .., \{m_1, m_2, .., m_{|R-methods|}\}\}$.*

NB: $M_{|R-methods|} = R-methods$ and $|M| = |R-methods| + 1$

Based on $Events$ ordered classification, Events Visibility Layers set structures process instance events in order to keep possible to negotiate a particular $Events$ subset (events layer) for process service interconnection:

Definition 11 (EV: Events Visibility Layers set) *We define Events Visibility Layers set EV as: $EV \subset \wp(Events)$ such that $EV = \cup_{i=0}^{|Events|}\{EV_i\}$ where $EV_i \subseteq (Events, \leq) = \{e_1, .., e_{|Events|}\}$, $EV_0 = \emptyset$, and $EV_{1\leq i\leq |Events|} = EV_{i-1} \cup \{e_i\}$. Thus, $EV = \{\{\}, \{e_1\}, \{e_1, e_2\}, .., \{e_1, e_2, .., e_{|Events|}\}\}$*

NB: $EV_{|Events|} = Events$ and $|EV| = |Events| + 1$

Example 5 Let us illustrate these concepts of visibility contract and process service API within our e-learning context : We know from M_i definitions that $R-methods = M_{|R-methods|}$ and $Events = EV_{|Events|}$. From examples 2 and 3, we have $R-methods = M_{12}$ and $Events = EV_5$, thus, in our example, we have $(M_{12}, EV_5) \subset$ Process API. Additionally, we have: $M_2 = \{$fetchProcessInstanceState(..), fetchActivityInstanceState(..)$\}$, and $EV_3 = \{$TerminatedProcessInstanceNotification, StartedProcessInstanceNotification, TerminatedActivityInstanceNotification$\}$. If R (the e-learning enterprise of example 1), negotiates, among those proposed, the visibility contract $V_i \in \overline{V}$ such that $V_i = (M_2, EV_3)$ for s_{11} (the internet site development process service) provided by P_1 (the web agency enterprise), then the process

[5] Let W a finite set, we note $\wp(W)$ the set of W subsets.

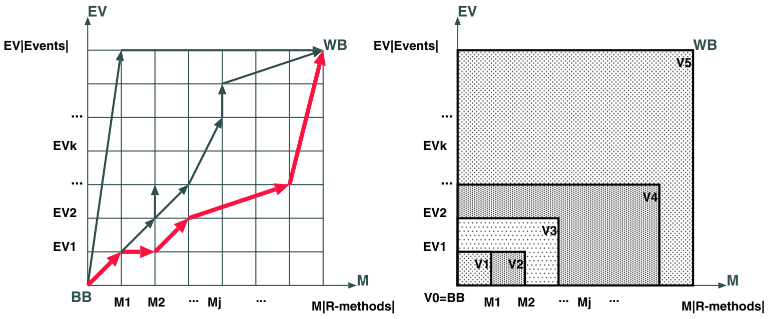


Fig. 3. Interconnection contract set

service API of s_{11} will be Process service API $= V_i = (M_2,\ EV_3)$. Knowing that $(M_2,\ EV_3) \subseteq (M_{12},\ EV_5) \subset$ Process API, we obtain then Process Service API $\subseteq$ Process API. In the sense of visibility, this means that Process Service API "hides" Process API as symbolised in figure 2.

As expressed previously, we aim to model a large process access rights panel offering possibilities between **black box (BB)** and **white box (WB)** (i.e. more flexible than black box but less permissive than white box). To enable process service requester and provider to co-decide easily which process service visibility contract to adopt during their process interconnection, we introduce the concept of Interconnection contract that structures Visibility contract set:

Definition 12 (I: Interconnection Contract set) *Let I the interconnection contract set. Each element of I is constituted of growing visibility contracts on process API methods and events. More formally, $I \subset \wp(\overline{V})$ such that:*

$$\begin{cases} I = \{i_j \subset \overline{V} \mid i_j = \{v_0, .., v_{p_j-1}\} \text{ with } |i_j| = p_j\} \ \wedge \\ v_0 = (M_0,\ EV_0) \in \overline{V}\ (\textbf{black box}) \ \wedge \\ \forall\ k \in \{0, .., p_j - 2\}, v_k, v_{k+1} \in \overline{V}^2,\ v_k \subseteq v_{k+1} \ \wedge \\ v_{p_j-1} = (M_{|R-Methods|},\ EV_{|Events|}) \in \overline{V}\ (\textbf{white box}) \end{cases}$$

Each $i_j \in I$ represents the alternatives set of visibility contracts the provider can propose for his service requester with p_j the number of these alternatives. Interconnection contract set can be applied as a tactical to insure visibility contract negotiation convergence (negotiation by conceding)[6].

While the left diagram shows some possibilities of interconnection contracts set establishment, the right diagram presents an element $i_j = \{v_0, .., v_{p_j-1}\}$ of I. This latter is the highlighted path in the left diagram, i_j is a path and each of its vertex is a visibility contract.

[6] Let $I = \{i_1, i_2, .., i_j\}$. The process service provider chooses $i^{'} = \{v_0, .., v_{|i^{'}|-1}\} \in I$ such that: if it proposed the visibility contract $v_t \in i^{'}$ (at the instant t) and the requester rejected it, the provider could propose $v_{t+1} \in i^{'}$ (directly succeeding v_t)

NB: Let $i_j \in I$, and $p_j = |i_j|$, we have $p_j \in \{2, .., |M| + |EV| - 1\}$.
In fact, p_j can reach at least 2
(e.g. $i_j = \{BB = (M_0, EV_0),\ WB = (M_{|R-methods|}, EV_{|Events|})\}$) and at best $|M| + |EV| - 1$ which is the maximum length of a path in $M \times EV$ grid.

We define the process service matching as the mechanism enabling the evaluation and comparison of process services. The process service profile is the key of this matching measures. Process service matching will improve specific process service awareness within business communities and will easy the process service accessibility and interconnection (e.g. process service web research engines, automatic process service evaluators, etc.). Matching techniques are necessary to help service requesters (or providers) to find the services that match their requested (or provided) services in the best way.

Definition 13 (*match***: Process service matching function)** *Let consider s_{req} and s_{prov} respectively the required and provided services, and let $dist : S \times S \rightarrow \mathcal{R}$ a distance function. We define a matching function as a predicate which verifies if both services match (i.e. the provided service s_{prov} suit the need of the requested service s_{req}), more formally:*

$$\begin{gathered} match(s_{req} \in S,\ s_{prov} \in S,\ \Delta \in \mathcal{R}) = true \Leftrightarrow \\ ((s_{req}.category = s_{prov}.category \vee \\ s_{req}.category\ is\_subtype\_of\ s_{prov}.category) \wedge \\ dist(s_{req},\ s_{prov}) \leq \Delta) \end{gathered}$$

In the sense of the distance function $dist$[7], the neighbourhood of a process service s is the set of all process services matching this process service in a "sphere" of radius Δ. This neighbourhood function is to be seen as a process service interconnection decision support mechanism rather than a research algorithm.

Definition 14 (*NH***: Process service neighbourhood function)** *Let $NH(s \in S,\ \Delta \in \mathcal{R})$ the neighbourhood of the process service s be the set of the services matching s and being near from it with a distance Δ. More formally: $NH(s,\ \Delta) = \{s^{'} \in S \mid match(s,\ s^{'},\ \Delta)\}$*

In order to define our process service model, we have firstly defined process service profiling concept (definition 7), then, we have defined process service API concept (definition 8). Definitions 9, 10, 11 described necessary concepts related to visibility contract notion. Definition 12 presented interconnection contract set concept. And finally, definitions 13, and 14 detailed concepts of process service matching and neighbourhood.

[7] The general form of the function $dist$ will be : $dist_n(s_1 \in S,\ s_2 \in S) = \sqrt[n]{\sum_{i=1}^{k} w_i\ |profile(s_1)_i - profile(s_2)_i|^n}$, where $(w_i)_{i=1,..,k}$ are positive real weights increasing with the importance we want to give to the i^{th} element of the profiles to be compared (for instance, with $(w_i)_{i\ =\ 1,..,k} = 1$ and $n = 1$, we obtain Hamming distance, and with $(w_i)_{i\ =\ 1,..,k} = 1$ and $n = 2$, we obtain Euclidean distance).

3.3 Process Service Interconnection Model

Having deeply developed our process model and process service model, we can define our process service interconnection model. We call process service interconnection, the possibility to link two enterprise processes by keeping them aware about requested and provided process services and enabling them to negotiate interconnection contracts before effective collaboration. This will be possible by deploying the process and process service model. Actually, to enable process service interconnection to be achieved, we will use the process service structure (profiling concept (definition 7), visibility contract concept (definition 9), process API concept (definition 8), etc.), and the algorithms manipulating this process service structure (interconnection contract set establishment (definition 12), distance measurement and matching predicate (definition 13), neighbourhood calculus (definition 14), etc.)

Additionally, we aimed to conceive a system to support dynamic co-operative process interconnection with neither previously determined communication primitives, nor previously scheduled points of rendezvous. For this purpose, negotiation mechanisms are necessary to support common decision making between processes (services to exchange, rendezvous, treaties, contracts, results needed, etc.). To enable these decision making processes to be insured, we will deploy our negotiation support system (NSS) [1]. It has been developed as generic as possible to permit negotiation support independently of any domain of application and of any negotiated items. Thus, it can be applied to co-decision problems between the service requester and service provider(s), involved in process service interconnection session.

We define process service interconnection methodology as the following operation sequence : Once defined (i.e. created and profiled) **(step 1)** then published **(step 2)**, a provided process service definition is accessible to all community enterprises. Each process service requester can retrieve process service definition, match it to its requested process service definitions and compute its process service neighbourhoods **(step 3)**. If the neighbourhood calculus returns interesting measures (according to requester point of view), it can initiate, with the process service providers, negotiation sessions to co-decide better process service profiles (in the sense of distance function) **(step 4)**. Once the process service profile is decided, the process service process service visibility contract has to be negotiated to permit the service API to be dynamically established **(step 5)**[8]. For two similar provided services, providers will have to offer the best profile and then the best visibility contract to conclude the service negotiation. Process service wrapping **(step 6)** is the final step of process service interconnection before process service enacting **(step 7)**. While process service wrapping deals with committing and dispatching agreed process service parameters (profile and visibility contract) on both process service views (the requester view and the provider view), process service enacting deals with enacting the process instance related to the process service. The exchanges between the process service requester and

[8] formally, it deals with selecting service visibility contract v among those already offered $v \in D = i_j \ (i_j \in I)$

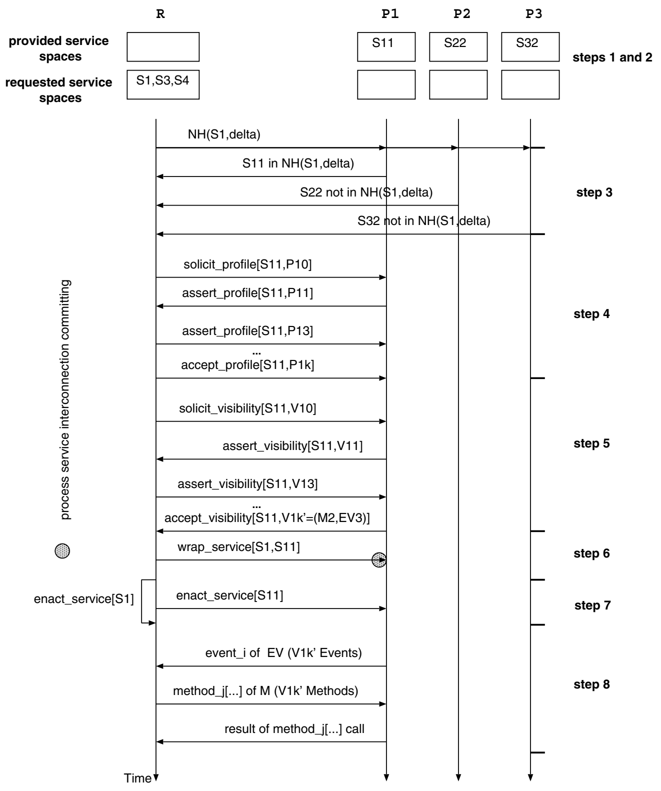


Fig. 4. Our process service interconnection example illustration

provider deal with the process service life cycle monitoring **(step 8)**. This concerns the use of the previously negotiated process service visibility contract (i.e. enacting process service methods, receiving process service events).

Example 6 We use the requested/provided e-learning portal development process services s_1/s_{11}, described in the examples 1, 4 and 5, to illustrate our process service interconnection approach through the description of the eight process service interconnection steps in terms of interactions between the process service requester and providers: Although implementation is not in the intended scope of this paper, we can mention that we have experimented our process service interconnection model as a process service exchange environment on our co-operative platform *DISCOBOLE* (DIStributed CO-operation and Business prOcess on LinE), the new generation of *DisCOO* [30], developed in Java on a CORBA bus. Among other concepts of the model, we have developed the process service profiling, the distance measurement, the matching predicate, and the neighbourhood calculus. Beside the defined scalar distance (see definition 13), we also considered its multidimensional form showing the distance elements on each process service profile attribute axis[9].

4 Conclusion and Perspectives

Business process interconnection becomes an important matter within Internet Economy. In fact, process interconnection mechanisms are necessary to strength awareness inside virtual enterprises, to facilitate electronic transactions, and to open the enterprise processes to Internet market places. Being developed to suit the intern needs of enterprises, most of existing process management systems are not adapted to inter-enterprise co-operation. Additionally, current models or solutions are mostly proprietary (i.e. based on : specific business process definition languages (BPDLs), particular WFMS platforms, private data exchange formats,...), or deal either with limited aspects of data communication protocols and make abstraction of process co-operation structure and semantics. Our paper is a contribution to the field of process service interconnection. It provides a process service framework for business process interconnection. We aimed mainly to treat certain aspects of co-operation "privacy" and "dynamics" compromises of business process interconnection problematic. Actually, in order to open co-operating processes without neglecting process privacy aspects, we developed the concepts of process service profiling and process service visibility contract. Additionally, we introduced mechanisms like process service matching and process service negotiation [1] to keep process interconnection flexible and dynamic rather than static and prescheduled.

Although our process service model is rich and flexible to support process interconnection, some complementary ideas and bricks have to be developed. Among other interesting works, we need to expand the visibility contract to process instance updating methods, to deep the concept of process service life cycle monitoring, and to treat the ideas of process service data space, process service quality, process service correction control, process service composing, and process service interconnection strategies.

References

1. M. Munier, K. Baïna, and K. Benali. A negotiation model for cscw. In O. Etzion/P. Scheuermann, editor, *5th Int. Conf. on Cooperative Information Systems (CoopIS'00)*, volume 1901 of *LNCS*, pages 224–235, Eilat, Israel, September 6-8 2000. 239, 250, 252
2. M. Reichert and P. Dadam. A framework for dynamic changes in workflow management systems. In *8th International Workshop on Database and Expert Systems Applications (DEXA'97)*, Toulouse, France, September 1-2 1997. 240
3. G. Alonso, D. Agrawal, A. El Abbadi, and C. Mohan. Functionality and limitations of current workflow management systems. *IEEE Expert - Special Issue on Cooperative Information Systems*, 12(5), 1997. 240

[9] $dist\_axis_{i \in \{1,..,k\}}(s_1 \in S,\ s_2 \in S) = w_i\ |profile(s_1)_i - profile(s_2)_i|$, with the same previously defined semantics of weights w_i as in definition 13

4. G. Alonso and C. Mohan. Workflow management systems: The next generation of distributed processing tools. In S. Jajodia/L. Kerschberg, editor, *Advanced Transaction Models and Architectures*, pages 35–62. Kluwer Academic Publishers, 1997. 240
5. A. P. Barros and A. H. M. Ter Hofstede. Modelling extensions for concurrent workflow coordination. In *4th IFCIS Int. Conf. on Cooperative Information Systems (CoopIS'99), IEEE Computer Society*, pages 336–347, Edinburgh, Scotland, September 1999. 240
6. C. Hagen and G. Alonso. Beyond the black box: Event-based inter-process communication in process support systems. In *19th International Conference on Distributed Computing Systems (ICDCS 99)*, Austin, Texas, USA, May/June 1999. 240
7. W. M. P. van der Aalst, P. Barthelmess, C. A. Ellis, and J. Wainer. Workflow modeling using proclets. In O. Etzion/Peter Scheuermann, editor, *5th Int. Conf. on Cooperative Information Systems (CoopIS'00)*, volume 1901 of *LNCS*, pages 198–209, Eilat, Israel, September 6-8 2000. 240
8. F. Casati and A. Discenza. Supporting workflow cooperation within and across organisations. In *15th ACM Symposium on Applied Computing (SAC'00)*, pages 19–21, Como, Italy, March 2000. 240
9. WFMC. *Workflow Standard - Interoperability, Abstract Specification, WFMC-TC-1012, Version 1.0*. WFMC (Workflow Management Coalition), www.wfmc.org, October 20 1996. 240
10. OMG. *Workflow Management Facility Convenience Document combining dtc/99-07-05 dtc/2000-02-03 (WF RTF 1.3 Report)*. OMG (Object Management Group), ww.omg.org, February 14 2000. 240
11. G. A. Bolcer and G. Kaiser. Swap: Leveraging the web to manage workflow (swap). In *IEEE Internet Computing*. www.ics.uci.edu/ ietfswap/, January-February 1999. 240
12. Y. Breitbart, A. Deacon, H.-J. Schek, A. Sheth, and G. Weikum. Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows. In *SIGMOD Record*, volume 22, pages 23–30, 1993. 240
13. D. Georgakopoulos, M. F. Hornick, F. Manola, M. L. Brodie, S. Heiler, F. Nayeri, and B. Hurwitz. An extended transaction environment for workflows in distributed object computing. In *IEEE Data Engineering Bulletin*, volume 16, pages 24–27, June 1993. 240
14. M. Rusinkiewicz and A. Sheth. Specification and execution of transactional workflows. In Won Kim, editor, *Modern Database Systems, The Object Model Interoperability and beyond*, pages 592–620. Addison Wesley, ACM Press, 1995. 240
15. D. Georgakopoulos, H. Schuster, A. Cichocki, and D. Baker. Managing process and service fusion in virtual enterprises. *Information Systems, Special Issue on Information Systems Support for Electronic Commerce*, 1999. 240
16. OMG. *Autonomous Decentralized Service System (ADSS) Domain Special Interest Group Whitepaper ver 1.0 (ads/97-12-01)*. OMG (Object Management Group), www.omg.org, December 5 1997. 240
17. G. Piccinelli. Distributed workflow management: The team model. In *3rd IFCIS Int. Conf. on Cooperative Information Systems (CoopIS'98), Sponsored by IFCIS, The Intn'l Foundation on Cooperative Information Systems*, pages 292–299, New York City, New York, USA, August 20-22 1998. IEEE-CS Press. 240
18. C. Godart, O. Perrin, and H. Skaf. coo : a workflow operator to improve cooperative modelling in virtual processes. In *9th International Workshop on Research Issues*

on Data Engineering : Information Technology For Virtual Enterprises (RIDE-VE'99), Sponsored by the IEEE Computer Society, Sydney, Australia, March 23-24 1999. 240

19. J. Klingemann, J. Wasch, and K. Aberer. Adaptative outsourcing in cross organizational workflows. In *11th International Conf. On advanced Information Systems Engineering (CaiSE'99)*, Heidelberg, Germany, June 14-18 1999. 240
20. F. Casati, S. Ilnicki, L. J. Jin, and M. C. Shan. eflow: an open, flexible, and configurable approach to service composition. In *Second International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS'00)*, pages 125–132, Milpitas, California, June 8-9 2000. 240
21. P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig. Crossflow: cross-organisational workflow management in dynamic virtual enterprises. In *International Journal of Computer Systems, Science and Engineering (IJCSSE'00)*, pages 277–290, 2000. 240
22. L. Kutvonen. *Trading services in open distributed environments*. PhD thesis, Department of Computer Science, University of Helsinki, Finland, 1998. 240
23. B. Benatallah, B. Medjahed, A. Boughettaya, A. Elmagarmid, and J. Beard. Composing and maintaining web-based virtual enterprises. In *VLDB workshop on Technologies for E-Services*, Cairo, Egypt, September 14-15 2000. 240
24. F. Leymann and D. Roller. *Production Workflow, Concepts and Techniques*. Prentice-Hall, Inc., 2000. 241, 242
25. WFMC. *Workflow Management Application Programming Interface (Interface 2 and 3) Specification, Document Number WFMC-TC-1009, Version 2.0*. WFMC (Workflow Management Coalition), www.wfmc.org, July 1998. 241
26. UDDI.Org, www.uddi.com. *Universal Description, Discovery and Integration (UDDI) Technical White Paper*, September 2000. 245
27. W3C, www.w3.org/TR/wsdl. *Web Service Description Language (WSDL) version 1.1*. 245
28. Microsoft, www.biztalk.org. *Microsoft BizTalk Server : BizTalk Framework 2.0: Document and Message Specification*, December 2000. 245
29. UN/CEFACT and OASIS, www.ebXML.org. *ebXML Technical Architecture Specification*, October 17 2000. 245
30. M. Munier, K. Benali, and C. Godart. Discoo, a really distributed system for cooperation. *Networking and Information Systems Journal*, 2(5-6):605–637, 1999. 251

Employing Multiuser Interactions in the Development of Synchronous Applications

Cornelia Haber

Carl von Ossietzky Universität Oldenburg, Fachbereich Informatik,
Escherweg 2, 26121 Oldenburg, Germany
haber@informatik.uni-oldenburg.de

Abstract. In the last few years cooperative systems have gained importance due to the fact that more and more people have access to networked computers. The technical, social and business impacts of using cooperative applications are researched but there are still uncertainties on how computers should ideally support cooperative work.
This paper suggests a methodology for the development of cooperative applications. As one very important question for developers of cooperative applications is how users interact with each other and with the application the methodology focuses on multiuser interactions. Multiuser interaction are interactions where multiple users work together in order to trigger the interaction or where users get feedback when the interaction is triggered by another user. A special model for multiuser interactions, MoMI, is introduced and its implementation discussed. The last part of the paper describes experiences in the development of a cooperative application using our methodology.

1 Introduction

With the growth of the Internet, at least one technical prerequisite for computer supported cooperative work is fulfilled. According to a study of the FZI in Karlsruhe [18] about 59% of the companies questioned in the area of information technology and biotechnology already use groupware and 98% use the Internet. However development of cooperative applications is expensive due to the complexity of the applications. Developing cooperative applications requires knowledge in distributed systems, software engineering, user cooperation, user interface design and so on. As different disciplines research cooperative applications and their impact on users it is important to find common languages and notations for describing the applications functionality, the user interface, the cooperation between users, etc. One such notation is MoMI, our **Mo**del for **M**ultiuser **I**nteractions. It is intended for multiuser interactions, i.e. interactions where more than one person either takes part in triggering the interaction or is affected by the interaction effects.

In this paper we describe MoMI, a methodology for the development of cooperative applications using MoMI and our experiences in using the methodology

© Springer-Verlag Berlin Heidelberg 2001

in the development process. The paper is structured as follows: The first section defines the application classes the methodology aims at. Then a process model heavily relying on MoMI, the multiuser interaction model, is described. The last part of the paper discusses the development of an application using the methodology. The paper closes with a short conclusion.

2 Target Applications

Various classifications for groupware exist distinguishing groupware according to the time and place the users take part in the application [4], the functionality of the application [15], or whether the application offers communication support, coordination support or cooperation support [16]. However none of these classifications is suitable to describe cooperative applications containing multiuser interactions as they do not stress the coupling of users.

In a first step we can restrict the applications our methodology aims at to synchronous applications as multiuser interactions only make sense if multiple users take part in the application at the same time. With reference to the MVC concept [1,14] we can distinguish synchronous applications where the users share a model and synchronous applications where the models of the users may differ. Applications with one model for all users may further be divided into shared applications, where all users have the same view of the application and applications where each user may have a view of his own. Different views do not only allow for small variations like the position of a window, they allow for completely different representations of the model data.

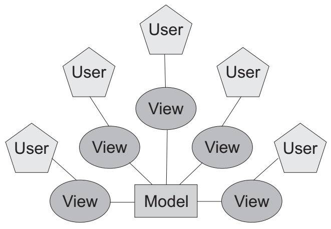


Fig. 1. One model many views

Our approach focuses on applications with one model for all users and any number of views as shown in Figure 1. In this application class users have to cooperate closely. They share data and often at least part of their view and therefore changes one user causes will quite often affect other users. Moreover interactions associated with more than one user will not be out of the ordinary in this application class. We will not consider applications with different models as there the coupling of users is quite weak. The users access different data, different data structures and their views of the application can be different. Therefore there are very few points of contact for the different users.

3 A Process Model for Cooperative Applications

The development process for a cooperative application consists of a number of phases the developer has to go through. In this section we describe a simple process model for the development of cooperative applications, closely related to the waterfall model. The phases of the process model are preconsideration, singleuser

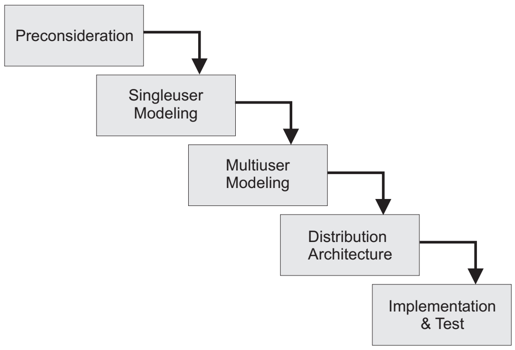


Fig. 2. Process model for cooperative applications

modeling, multiuser modeling, distribution architecture and implementation & test as shown in Figure 2.

3.1 Preconsiderations

Cooperative applications vary in the way the users cooperate and in the coupling of users. These aspects have to be considered before the application can be modeled in detail. The preconsiderations for cooperative applications can be divided in two main groups:

– User aspects
– Network aspects

As multiple users take part in a cooperative application some questions concerning the number of users and the ways users cooperate with the application and between themselves are raised. Modeling a cooperative application the first question is whether the application should be a synchronous one where the users use the application at the same time or if it should be asynchronous, i.e. an application where the different users take part in the application at different times [4]. If the application is synchronous the application designer has to decide whether the application is to be collaboration transparent or collaboration aware. An application is collaboration aware when it was designed as a multiuser application. Collaboration transparent applications are applications designed as singleuser applications which are made multiuser compliant with the help of tools like shared application systems [12].

After the application designer decided on the cooperation between users he has to take a close look into the users. How many users shall take part in the application? Is the number of users variable within the application? Do all users have to take part from the beginning or may users join into the application later? Those considerations have an impact on the application design as for example for latecomers the actual status of the whole application has to be provided.

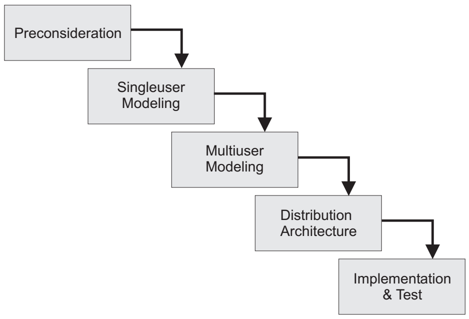


Fig. 3. Multiple user considerations

The last group of questions concerning users deals with the discrimination of users. Shall the application distinguish between users? Has the discrimination of users an impact on the application flow, i.e. different users may have different application flows, or is the discrimination just important for access rights? In order to distinguish users it may be sensible to introduce groups and roles. Regarding groups and roles new questions arise on the group/role structure and whether the group and role structures are static or dynamic. Figure 3 summarizes the main questions concerning users.

Network aspects take the distance between the users into account. Are the users taking part in the application connected via a local area network (LAN) or via a wide area network (WAN). Not the actual distance in kilometers is important but the network transmission rates. Do all users have similar rates? Depending on the transmission rates fairness strategies have to be integrated to ensure similar conditions for all users.

After the preconsiderations are finished, the designer should model the different views on the application, i.e. the way the application presents itself to the different users/groups/roles.

3.2 Singleuser Modeling

Singleuser Modeling in our methodology is synonym to modeling the application as it is seen through the eyes of a user. This is a task well known in software engineering. Therefore existing methods, notations and tools (e.g. UML) can be used. If the application flow differs for different users/groups/roles one model per user/group/role has to be developed. The models need not be detailed. They just build the basis on which to discuss the application and the functionality it offers the users at each time. UML activity diagrams [6] are one way to represent these coarse models. They focus on activities and the sequencing of activities. Activity diagrams allow for parallel activities and their synchronization.

When the application seen from one user's view is modeled the other users are neglected. One reason for this simplified version is that the model is smaller and therefore easier to read and understand. The other reason is that this model shows what the user will find using the application. For the user there will be no difference whether another user causes a change in the application or the application itself causes the change. Both times the user does not trigger the change himself.

The singleuser model has to contain the following information:

- Where and when the user may influence the application flow through user interactions. At this point singleuser as well as multiuser interactions have to be considered. So far it is sufficient to know that the user can take part in a multiuser interaction, the interaction itself is not specified in detail.
- Application flows determined through the application itself or other users.

When the different singleuser models of the application are settled they have to be integrated and made a multiuser model.

3.3 Multiuser Modeling

The next step in the development process is to enhance and combine the singleuser models in order to get a multiuser model. Therefore the correlations between the different models have to be identified.

Correlations between Singleuser Models The correlations between the different models show where the users may influence each other, which constraints may be between the applications at the different user's sites and so on. There are two main ways users may influence each other:

Causal connection: A causal connection between two models exists when an activity in one model has to be fulfilled in order to enable another activity in the other model. This is an analogon to a simple sequence in the activities in one model only that the causal connection is a sequencing of activities in different models.

Corporate interactions: Corporate interactions are interactions where different users have to take part in the interaction, i.e. they each have to trigger their part of the interaction. A corporate interaction may be triggered by different users causing the same action or by different users causing different actions. The main difference between corporate and multiuser interactions is that in corporate interactions multiple users have to trigger the interaction while in multiuser interactions it is sufficient for one user to trigger the interaction as long as the interaction affects multiple users.This way every corporate interaction is a multiuser interaction as well.

After the correlations are identified they have to be analyzed, specified in detail and integrated into the combined model. Causal connections can be represented through arrows sequencing activities in an activity diagram. Corporate interactions have to be specified using MoMI.

Multiuser Interactions Multiuser interactions are interactions where different people take part in the interaction, i.e. either different people work together to trigger the interaction (e.g. by each user pressing a button) or different users are affected by the interaction. The main questions regarding multiuser interactions are which users/groups/roles take part in the interaction, which order do the different users/groups/roles have to act in and are there any time restrictions for the interactions?

In the field of multimedia an interaction is defined mainly as the way a user may influence the application flow [3]. An interaction is characterized by interaction task, interaction technique, interaction form, interaction device and the effect of the interaction. According to [5] an interaction task is the entry of a unit of information by the user. The interaction tasks classify the fundamental types of information entered with the interaction techniques. Interaction tasks are defined by what the user accomplishes. Interaction forms are the software part that inputs dates into the computer (e.g. masks and user interface components) [3]. Interaction techniques are ways to use input devices to enter information into the computer where input devices are computer hardware like mouse and keyboard. The interaction effect specifies the reaction the user expects from the application.

The characteristics of multiuser interactions are quite similar to singleuser applications but there are some differences. Whereas the interaction tasks in singleuser applications are select, position, quantify and text in multiuser interactions they are multiuser select, multiuser position, multiuser quantify and multiuser text. In a multiuser interaction it may not be sufficient for one user to select an object (task: select) it may be required that a given number of users have to select the object (task: multiuser select). The interaction task may even be the communication between users. Interaction techniques, forms and devices in multiuser interactions are the same as in singleuser interactions. Whereas the effect of the interaction in singleuser interactions is only relevant locally for the user triggering the interaction it may be relevant globally (to other users) in multiuser interactions. One multiuser interaction may even have different effects for different users. For the specification of multiuser interactions a special model, MoMI, is introduced.

Modeling Multiuser Interactions In MoMI, our model for multiuser interactions, an interaction can be seen as a kind of Event Condition Action (ECA) rule. On the occurrence of an event an action is triggered if the condition evaluates to true. In this context the event may be a complex event, consisting of a number of base events like a mouse click (first mouse down then mouse up). A simple multiuser interaction is then defined as an interaction where all users causing the interaction trigger the same event. A multiuser interaction in MoMI is characterized as follows:

Event: The (complex) event that is expected by the interaction. After the event occurs the condition is evaluated.

Condition: A boolean expression on values of attributes. A condition compares an attribute to a value. Conditions can be combined to complex conditions using boolean operators.

User: This field restricts the users who may cause the event. The restriction may be on the users identity, their groups or their roles. Moreover the number of users to cause the event is given in this field. The whole specification may then look like "5 users of group pupil".

Action: The action specifies the reaction of the application (local and global) that is triggered if the condition is evaluated to true.

Feedback: Direct feedback to the user causing the event.

Feedthrough: Direct feedback to all users except the one causing the event telling them that the event occurred.

Time restriction: Restriction on how long the interaction object shall wait for the events to occur. The time may be counted from the time the first event relevant for the interaction took place or from the time the multiuser interaction started (the time the interaction started to wait for events).

Reaction on abortion: When an interaction is aborted because the time exceeded the reaction of the application has to be specified. There are three basic reactions on an abortion. The first is to reinitialize the interaction so that the users may start again with this interaction. The second one is to drop the interaction and proceed with the application at a spot especially designed for this situation. The third possibility is to drop the interaction and proceed as if the interaction would not exist at all.

In specifying a simple multiuser interaction not all characteristics need to be given. Mandatory are the event and action. All other characteristics are optional.

Multiuser interactions may be combined to complex multiuser interactions using logical operators. MoMI supports the three operators *AND*, *OR* and *SEQ*. Combining multiuser interactions with the AND operator the interaction takes place if *all* part-interactions took place. The OR operator expects *one* part-interaction to take place and the SEQ operator is fulfilled if *all* part-interactions took place *in the order specified*. Logical operators may be used to combine simple and/or complex multiuser interactions. Together with the logical operators time restrictions, feedback and a reaction on abortion may be specified for the interaction described by the logical operator. As a number of users take part in a multiuser interaction it does not make sense to specify the user causing the interaction. Therefore feedback specified with a logical operator concerns all users (= feedback + feedthrough in simple multiuser interactions).

A multiuser interaction can be modeled as an interaction tree where the leafs are simple multiuser interactions (multiuser interactions where all users have to trigger the same action) and the inner nodes are logical operators. More information about MoMI can be found in [7].

Once the singleuser models are combined to a multiuser model and the multiuser interactions are specified the application developer has to decide on the architecture of the application.

3.4 Distribution Architecture

As each multiuser application is inherently a distributed application, some consideration need to be given to the architecture of the application and the distribution of the data. We distinguish three different ways to distribute the application:

– centralized architecture
– replicated architecture
– hybrid architecture

A centralized architecture consists of a central component, where the application is running and clients that are responsible for displaying the application to the users. User inputs are sent to the central component, handled there and all changes to the display are propagated to the clients.

The replicated architecture is the other extreme. There all users run their own instance of the application. Each instance contains all data structures, propagates changes of the data to the other instances and handles all user input.

The hybrid architecture combines the advantages of centralized and replicated architecture. With a hybrid architecture part of the data structures is replicated so that it may be manipulated locally. Changes relevant to other users need to be propagated via the server or directly to all other users applications. Figure 4 shows an example of a hybrid architecture.

Today most distributed applications have a hybrid architecture but they differ in the amount of data that is replicated and in the functionality the central component provides. There really is a wide range of hybrid architectures from close to centralized architectures to nearly replicated ones.

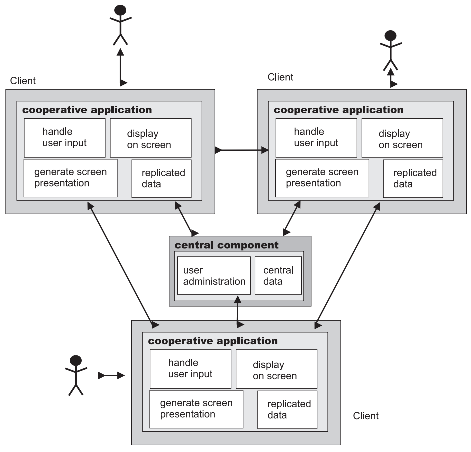


Fig. 4. Hybrid architecture

3.5 Implementation and Test

After the application design is finished the application needs to be implemented and tested. There is a variety of tools to support the implementation of cooperative applications from shared object systems ([8,17]) and multiuser user interface tools ([2]) to collaboration platforms ([13]) and frameworks ([9,10,11]). Depending on the general considerations the developer has to decide whether or not one of these tools is suitable for the implementation.

For the implementation of multiuser interactions we developed a framework implementing MoMI - the MoMI-Framework. The framework is written in the programming language java. A class diagram showing the main classes of the framework is shown in figure 5.

As the names suggest the classes AndKnoten, OrKnoten and SeqKnoten are the implementations of and-, or-, and seq-nodes in the interaction tree. The class InnererKnoten is an abstract class realizing inner nodes in the interaction tree and the leaves of the tree, i.e. simple multiuser interactions, are implemented in the class Blatt. Blatt and InnererKnoten implement the interface Knoten, which stands for all nodes of the tree.

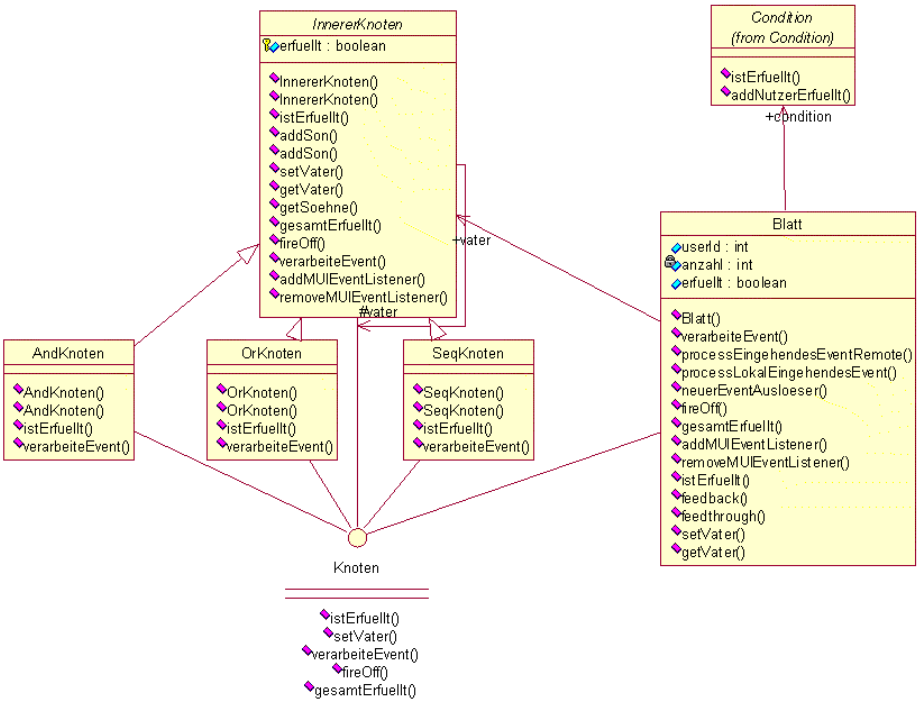


Fig. 5. Class diagram of the MoMI-Framework

During the implementation and after the code is written it has to be tested. This is another well known task in software engineering. Special to testing of cooperative applications is the testing of the cooperation between users.

As the discussion of implementation and testing of cooperative applications is a whole subject on its own it will not be discussed further in this paper.

4 Experiences with the Methodology

As a proof of concept, the in Germany familiar game "Mensch ärgere Dich nicht", a game like Parcheesi or Aggravation was modeled and implemented using our methodology. This section describes the work done in the different development phases.

4.1 Preconsiderations

The general considerations concern the users and the distribution of the users. First let's have a look into the users.

The game is played synchronously with two to four players. All players have to take part from the beginning (before the colors are chosen). Joining later into the game is not feasible and players may not leave the game. The users are distinguished against each other (each user may only play his/her own pawns) but the application flow is the same for all users.

There are no constraints regarding the network connectivity of the users. They may be connected over a LAN or a WAN or just be anywhere on the Internet.

4.2 Singleuser Modeling

The application roughly consists of three phases: The joining of the users in the application together with their introduction, the choosing of a color/role for each user and then the game itself (play).

First the user has to join the application, registering with the application and giving a nickname for himself. Then each user has to choose his color for the game whereby each user has to choose a different color, associating him with his pawns. The last activity – play – is more complicated and may be split up in rolling the dice and then (if possible) moving one pawn accordingly. Then the user has to wait for all other users to take their turn in the game before he can start his turn again by rolling the dice. Considering all part activities the application presents itself to a singleuser as shown in figure 6.

Once the application from one user's view is modeled, dependencies between the different users have to be identified.

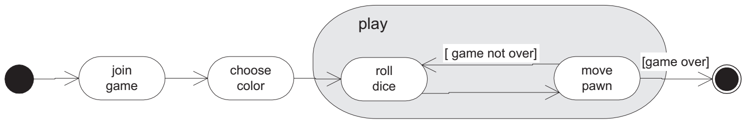


Fig. 6. Singleuser model of the application

4.3 Multiuser Modeling

In order to make the singleuser model multiuser compliant activities have to be identified, where the users influence each other, i.e. where the actions of one user influence other users or where multiple users are needed to fulfill an interaction. In our example, all activities concern more than just one user. Let's go through them one by one. The first activity is for a user to join the game. In doing so, he will give a nickname to introduce himself to the other users. The activity itself is initiated by one user and no other users take part in it, but the other users have to be informed of the existence of a new player. Therefore the interaction can be modeled as a singleuser interaction. The interaction effect would then cause e.g. a message to all users telling them that a new user has joined the game.

The second activity is for each user to choose a color. It is not sufficient to let each user choose a color and then tell all other users which color he chose as all users have to have different colors. A choice element is needed that ensures that every user chooses one color and that the other users choose different colors. In our example we modeled the choosing of a color as a multiuser interaction. Figure 7(b) shows the multiuser interaction "color choosing".

For each color to choose (the number of colors is the same as the number of users, in this example it is four) a simple multiuser interaction is defined. Figure 7(a) shows the details of the simple multiuser interaction.

The simple multiuser interaction "choice color x" is fulfilled if one user chooses the color. Next it has to be ensured that no other user chooses the same color. This is realized with the help of feedthrough. As feedthrough the choice for the color chosen is disabled for all users. Then it has to be ensured that each users makes just one choice for a color. Therefore as a feedback all choices

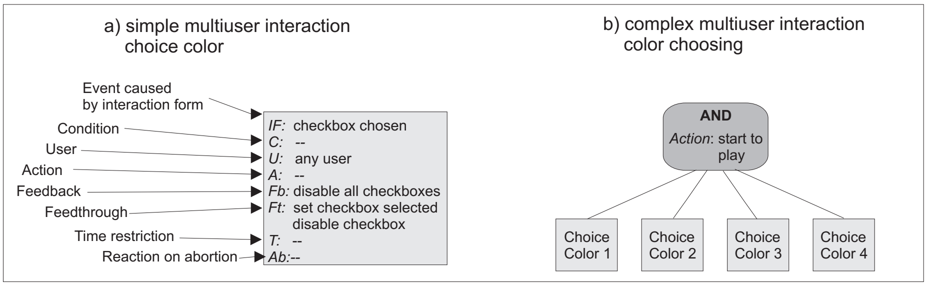


Fig. 7. Color choosing

are disabled for the user who has just chosen a color. Linking all color choices with the logical operator AND makes sure, that the color choosing interaction is finished when all users have made their choice.

The third activity is playing the game, i.e. for each player alternately to roll the dice and move a pawn until the game is finished. Only one user is active in the activities roll dice and move pawn, but all other users have to be informed on the outcome of the activity (the number after rolling the dice and the pawn the user moved). This can be modeled quite easily as a multiuser interaction as figure 8 shows.

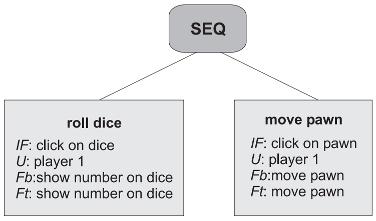


Fig. 8. Activity play

The sequence (SEQ) ensures that the activities are taken in the right order. It has to be specified that just the one user is allowed to roll the dice and to move a pawn whose turn it is. This is realized by restricting the simple multiuser interactions (roll dice and move pawn) to this user (in our example player 1). All other users actions are then ignored. Now we have to inform all other users that the simple multiuser interaction is fulfilled. Again this is done with the help of feedback and feedthrough.

At this point we need not distinguish between feedback and feedthrough because the application shall give the same reaction to all users on the occurrence of the interaction. Therefore as feedback and feedthrough for the simple multiuser interaction "roll dice" the number on the dice is shown and as feedback and feedthrough on the "move pawn" interaction the pawn is moved on all users screens.

Combining the singleuser models the multiuser model shown in Figure 9 emerges for four players. The activities *choose color player 1-4* are highlighted in grey to stress their being corporate interactions. The causal connections specifying the turns the users take in rolling the dice and moving a pawn are realized through a sequencing of the activities.

Now that the multiuser model is finished the application has to be implemented.

4.4 Implementation and Test

As architecture for the game we choose a hybrid architecture where the central component administrates the users. The rest of the data – field, pawns, dice, etc.

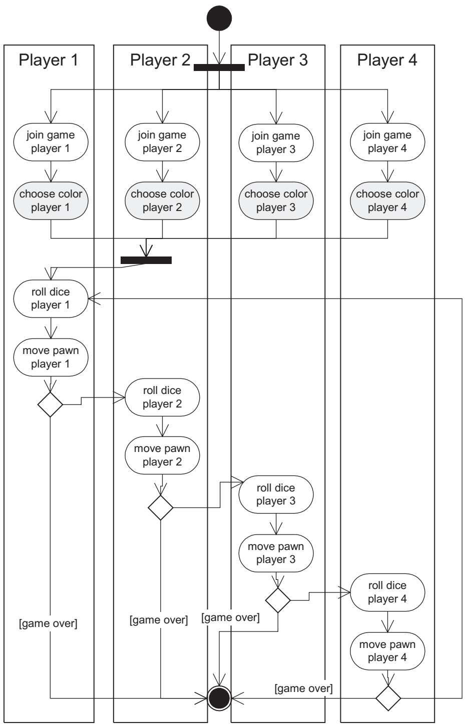


Fig. 9. Multiuser model

– are implemented as replicated objects. The whole application is developed in the programming language Java.

All multiuser interactions including interactions triggered by one user causing a feedthrough to other users are implemented with the help of the MoMI-Framework. Figure 10 shows the GUI element realizing the color choosing. It was implemented using checkboxes. Checkboxes selected by a user show the name of the player who selected the box and therefore the color.

The board is shown in figure 11.

Once the dependencies between users were defined the development of the game did not take long.

5 Conclusion

The development of the game has shown that using a model for multiuser interactions can make developing cooperative applications easier. The model should be used, when the dependencies between users interactions are complex or when more than one user take actively part in the interaction.

The implementation of MoMI is still a prototype and there are some deficits concerning performance and latecomers right now. Moreover tools are needed to

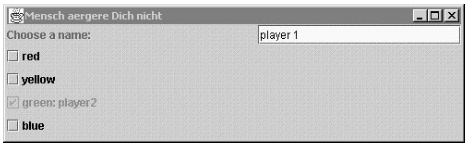


Fig. 10. Color choosing

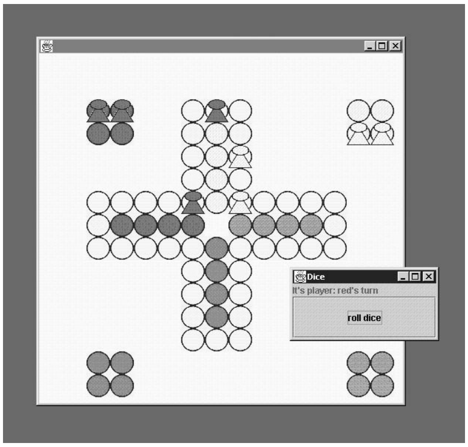


Fig. 11. Board

support developers using the model like an editor for modeling multiuser interactions. This editor should then generate the code for the multiuser interactions. However the prototype has proofed that the concept of multiuser interactions helps understanding and developing cooperative applications.

The methodology for the development of cooperative applications is feasible for synchronous applications with one model and multiple views.

References

1. Ayers Kenneth E., The Mvc Paradigm in Smalltalk/V. Dr. Dobbs Journal, 1990. 256
2. Begole James Bo, Struble Craig A., Shaffer Clifford A., and Smith Randall B. Transparent Sharing of Java Applets: A Replicated Approach. In Symposium on User Interface Software and Technology (UIST'97), pages 55–64. ACM Press, 1997. 263
3. Boles Dietrich. Das IMRA-Modell - Integration von Interaktionen in das Autorenwerkzeug FMAD. thesis, Carl von Ossietzky Universität Oldenburg. 260
4. Ellis Clarence A., Gibbs Simon J., and Rein Gail. Groupware: some issues and experiences. Comm. of the ACM, 34(1):49–58, 1991. 256, 257
5. Foley James D., van Dam Andries, Feiner Steven K., and Hughes John F. Computer Graphics: Principles and Practice. Addison-Wesley, 2nd edition edition, 1998. 260

6. Fowler Martin and Scott Kendall. UML konzentriert. Addison-Wesley, 1998. 258
7. Haber Cornelia. Modeling multi-user interactions. In Dillenbourg P. et al, editor, European conf. on computer supported cooperative learning, pages 277–284, Maastricht, Netherlands, 2001. Mc Luhan institute. 261
8. Huebner John and Myers Brad A. Easily Programmable Shared Objects For Peer-To-Peer Distributed Applications. Technical report, Carnegie Mellon University. 263
9. Jackson Larry S. Java Collaborative Technology Selections in NCSA Habanero. In 2nd Int. Conf. on Concurrent Engineering in Construction – Challenges for the New Millennium, pages 37–46, Espoo, Finland, 1999. CIB. 263
10. Jackson Larry S. and Grossman Ed. Integration of Synchronous and Asynchronous Collaboration Activities. ACM Computing Surveys, 31(2es), 1999. 263
11. Johnson Philip. Egret: A Framework for Advanced CSCW Applications. ACM Software Engineering Notes, 21(2), 1996. 263
12. Lauwers Chris J. and Lantz Keith A. Collaboration awareness in support of collaboration transparency: requirements for the next generation of shared window systems. In Conference onHumanFactors and Computing Systems, pages 303–311, Seattle, WA USA, 1990. 257
13. Roth Jörg. DreamTeam - A Platform for Synchronous Collaborative Applications. AI & Society, 14:98–119, 2000. 263
14. Shan Yen-Ping. MoDE: a UIMS for Smalltalk. In Conference on Object Oriented Programming Systems Languages and Aplications, pages 258–268, Ottawa, Canada, 1990. 256
15. ter Hofte G. H. Working apart together: Foundations for component groupware, volume 001. Telematica Instituut, Enschede, Netherlands. 256
16. Teufel Stephanie, Sauter Christian, Mühlherr Thomas, and Bauknecht Kurt. Computerunterstützung für die Gruppenarbeit. Addison-Wesley, Bonn, 1995. 256
17. Trevor Jonathan, Rodden Tom, and Mariani John. The use of adapters to support cooperative sharing. In Computer Supported Cooperative Work, pages 219–230, Chapel Hill, USA, 1994. 263
18. Weiss Peter. Studie Virtuelle Unternehmen. Technical report, FZI Forschungszentrum Informatik, 09 2000. 255

Service Representation, Discovery, and Composition for E-marketplaces

Willem-Jan van den Heuvel, Jian Yang, and Mike P. Papazoglou

University of Tilburg, Infolab
PO Box 90153, 5000 LE, Tilburg, Netherlands
{wjheuvel,jian,mikep}@kub.nl

Abstract. Although network-centric technologies will make the diverse services easily accessible via the Web, the discovery of the right service with the right capabilities and the development of E-commerce service based application and networked services which share existing e-services is still an ad-hoc, very demanding, and time consuming task. In this paper we propose a framework for e-services description and steps for service discovery and combination. Several algorithms for service discovery are proposed, which provide a foundation for service combination in an e-marketplace.

1 Introduction

The Internet now makes it possible for people to allocate services and build complex applications by combining data and processes offered by different services available across the network. By e-service, we refer to self-contained, Internet-enabled applications capable not only of performing business activities on their own, but also possessing the ability to engage other e-services in order to complete higher-order business transactions. Examples of such services include ordering products, making payment, checking order status, scheduling shipmentst and so on.

Each provided e-service contains a service interface and a service description and may be composed of sub-services. E-services can continue to recur until an atomic service is reached. At the highest level, businesses interact by using the services of each other within the context of some business process. The development of cooperative applications which share Web-accessible services is still an ad hoc very demanding, and time consuming task. It typically requires a tedious effort of low level programming. The existing tools provide little help in organizing service space which makes the efficient use of the services extremely difficult. The difficulty is threefold: (1) the service space is normally large and highly dynamic; (2) there is a lack of underlying framework to describe and present service in a way that they can be effectively discovery and accessed; (3) there are usually multiple services which offer similar things with some variation (e.g., different parameters, different access interface, different costs, etc).

Our goal for service discovery is to find the right service(s) with the right capabilities so that service invocation can be correctly and service combination

© Springer-Verlag Berlin Heidelberg 2001

can ensue. This will enable organizations to adapt their business practices to the highly dynamic nature of the web, and as a result the concept of virtual enterprise becomes a reality.

The remainder of the article is organized as follows. In section 2, we will discuss how the existing technologies have contributed in the area of service description and discovery, and their limitation. In section 3, we will firstly present a motivating example and then outline several challenges for developing a framework for service presentation and discovery. Section 4 discusses our approach for service description which provides a foundation for service discovery and combination. Section 5 presents a service hierarchy and discusses how to derive service capability based on the service hierarchy. Section 6 describes the steps in service discovery and a set of algorithms are developed accordingly. Service combination based on interface checking is explained in section 7. We conclude our article with a summary in section 8.

2 Related Work

There are quite a few activities in the area of e-services development, discovery, and combination. This is particularly true for XML standards proposed for specifying e-service interfaces and their registry procedure and repository.

UDDI is a joint initiative from IBM, Microsoft and Ariba and aims at an XML schema that supports a SOAP based protocol to publish and discover services on the web. UDDI basically offers programmers the posibility to compose E-service applications by selecting and composing E-services from the UDDI Business Registry. The registry therefore stores three types of information: business contact information (”white pages”), business category information, e.g., location, geographic region and category(”yellow pages”), technical service information, e.g., binding information and network protocol specifications (”green pages”) [10].

IBM has developed an XML-based service description language, called the Web Service Description Language (WSDL) to represent message content independently from the network protocol binding. At this moment, bindings have been described to apply WSDL in conjunction with SOAP 1.1, HTTP GET/ POST, and MIME [5]. IBM is currently developing an extension to Websphere, its solution to build electronic marketplaces, so it can deal with UDDI-based e-service architectures [6].

The other relevant initiatives in e-service development is e-speak (www.e-speak.net) from Hewlett Packard (www.hp.com) which is an open software platform designed specifically for the development, deployment, and intelligent interaction of e-services. E-speak allows the clients/service to interact with each other in a more abstract way. Once an E-service is e-speak-enabled, the provider has to register the service with a host system by creating a description of the service that consists of its specific attributes.

The workflow community has recently paid attention to configurable or extensible workflow systems which present certain overlaps with the ideas reported in the above. For example, work on flexible workflows has focused on dynamic process modification [8]. In this publication workflow changes are specified by transformation rules composed of a source schema, a destination schema and of conditions. The workflow system checks for parts of the process that are isomorphic with the source schema and replaces them with the destination schema for all instances for which the conditions are satisfied. They also propose a migration language for managing instance-specific migrations.

The approach described in [3] allows for automatic process adaption. The authors present a workflow model that contains a placeholder activity, which is an abstract activity replaced at run-time with a concrete activity type. This concrete activity must have the same input and output parameter types as those defined as part of the placeholder. In addition, the model allows to specify a selection policy to indicate which activity should be executed.

An interesting approach is that described in [1] where the authors discuss the development of a platform specifying and enacting composite services in the context of a workflow engine. The eFlow system provides a number of features that support service specification and management, including a simple composition language, events and exception handling. A traditional transactional support is also supported for processes that need to be executed in an atomic fashion. eFlow provides an open and dynamic approach to service selection.

The current e-service research activities suffer from various shortcomings:

– State of the art service representation languages are not able to sufficiently capture the semantics of the business domain and the structure of the service (e.g., the sub-services, the parts of the services);
– Business process dynamics are only partially covered by current service descriptions in terms of operations (capabilities): business dynamics and policies (constraints) are lacking. Business protocols only have been treated in cursory manner thus far.
– Service descriptions are not based on a solid type system. This puts a severe barrier on composing e-services dynamically, as the conformance of the resulting e-service suite can not be checked at runtime.
– E-service searching is only based on attribute match which is not sufficient enough to deal with partial matches and semantic conflicts.

Our goal is to extend the existing work by providing a comprehensive framework for service description and strategy for effective service discovery and combination.

3 Running Example and Issues

In this section we will outline the issues in service description and discovery by firstly introducing a motivating example which will be used throughout the paper. Assume that the entire buying process of a second hand car requires the following four main e-services:

– Search for a car based on the "make" and "type" of the car.
– Negotiate about the terms of delivery, e.g., price for the car, guarantee on the car, price for delivery, and so on.
– Order a car, which is composed of four part e-services: provide car information, provide contact information, provide payment information, provide shipment information.
– Delivery and payment.

We assume that there are several car sell services available on the web. Now the challenge is to represent the requested services and the available e-services so that the service match making can be made and the most appropriate service can be found.

In this paper we address the following issues related to services description

– The domain model used to describe "what is the service about". For our example, the domain is about `car sell`.
– The service "capabilities" indicate the structure of the service, i.e., the components of the service.
– The access syntax for using the service, e.g, how to invoke the service, the parameters, the order of the invocation of the components . This information is the important basis to determine service composibility, compatibility, conformance, and substitutability for service composition.

For the service discovery, the importance is to find the right service with the right capability. In case that multiple services are found, some criteria need to be developed to select the "best" service. Sometimes several services need to be combined to accomplished the requested service, the syntax of the relevant part services need to be checked to decide if they can be combined. Thus we have developed the following steps for service discovery:

– Semantic relatedness: at this step, the requested service is compared against the service description in the repository in terms of service contents to decide how close they are. These services with a high degree of relatedness will be selected as relevant services for further capability checking.
– Capability analysis: for the relevant services selected from the previous step, we check their capabilities in terms of the provided functions to determine whether they can accomplish completely or partially. the desired tasks in the requested service.
– Syntactic analysis: For those "capable" services, we shall check the syntax of the interface to determine how they can be combined to achieve the higher-order requested service.

4 Service Representation

We have developed a service description language (SDL) to address the above issues. In the next two sub-sections, we shall first explain the structure of SDL, then we shall illustrate the syntax of SDL in terms of an XML document using the running example.

4.1 The Structure of SDL

The Service Definition Language (SDL) we propose is expressive enough to describe:

- *Service properties*, i.e, service general information (e.g., identification, owner), service access information (e.g., service location - URL, the maximum time for a conversation between the service and a service requester, public key certificate), and contact information.
- *Service Ontology*, i.e, what is the service about and the terminology that is needed to discover the service.
- *Service Cost*, i.e, the estimated cost for using the service or the information provided by the service.
- *Payment*, i.e, the way the service receives the payment from the customers.
- *Actors*, i.e, the people or organizations who are using the service.
- *Authorization/security/visibility*, i.e, who can see/use what (service contents and functions).
- *Service contents*, which specifies the content and the structure of the underling service, e.g., the attributes, objects, the constraints on use of attributes/objects, etc.
- *Service capability*, which specifies the service structure/components, the conditions of using the service, and the order of component invocation.

In the next session, we only concentrate on the service *properties, ontology, contents,* and *capability* description.

4.2 SDL Documents: An Example

The requested service and the offerring service `car_sell` presented in Section 2 can be described in SDL as depicted in Figure 1 and Figure 2, respectively.

```
<!--*********** Specification of Request ********************* -->
<request>
  <from src="http://www.infolab.nl/jian />
  <vocabulary name="second hand car dealer" />
  <service name="sell car" />
  <service name="search car" />
  <service name="Negotiate", option="optional" />
  <service name="order car" />
  <service name="fulfillment" />
  <result> $serviceInfo </result>
</request>
```

Fig. 1. A Requested Service

SDL documents on one hand, need to be accessible to the end-users without the technical details such as network and security information. On the other hand, programmers who are responsible for e-service integration require these

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE SDL_definition SYSTEM "sdl3.5.dtd">
<!-- GREEN PAGE VIEW ON THE SERVICE DEFINITION  -->
<!-- ************Specification of the car seller ************* -->
<SDL_definition>
  <!-- This e-service implements two variants of the "search_car" method-->
  <service name="search_car" from="http://www.cardealer.nl">
    <provides_service_operation>
    <option>
      <operation name="search_car">
        <input_parameter type="string">make</input_parameter>
        <output_parameter type="business_object" business_object_ref="carinformation">carinformation</output_parameter>
      </operation>
    </option>
    <option>
      <operation name="search_car">
        <input_parameter type="bag">type</input_parameter>
        <output_parameter type="business_object" business_object_ref="carinformation">carinformation</output_parameter>
      </operation>
    </option>
    </provides_service_operation>
    <!-- To garantuee type safe orchestration of e-services, we now specify the business object data type -->
    <business_object name="carInformation" visibility="private">
      <attribute att_name="brand" att_type="string"/>
      <attribute att_name="model" att_type="string"/>
      <attribute att_name="convertible" att_type="boolean"/>
      <attribute att_name="color" att_type="string"/>
      <attribute att_name="mileage" att_type="integer"/>
      <attribute att_name="nr_of_doors" att_type="integer"/>
    </business_object>
  </service>

  <!-- The service specification for the higher-order service "Order Car"-->
  <service name="Order_Car" from="http://www.carmarket.nl">
    <service name="provide_car_information" from="http://www.cardealer.nl/">
    <!-- The requires interface specification for the provide_car_information e-service  -->
      <uses_service_operation>
        <operation name="search_car">
          <input_parameter type="string">type</input_parameter>
          <input_parameter type="integer">customerID</input_parameter>
          <output_parameter type="business_object" business_object_ref="carinformation">carinformation</output_parameter>
      </operation>
        <operation name="getPriceInformation">
          <input_parameter type="string">type</input_parameter>
          <input_parameter type="bag">model</input_parameter>
          <input_parameter type="integer">year</input_parameter>
          <output_parameter type="integer">priceOfCar</output_parameter>
      </operation>
      </uses_service_operation>
      <!-- Provide car information part-->
      <provides_service_operation>
        <operation name="provide_car_information">
          <input_parameter type="string">confirmation_of_purchasing</input_parameter>
          <output_parameter type="string">carInformation</output_parameter>
      </operation>
      </provides_service_operation>
      <business_object name="carInformation" visibility="private">
        <attribute att_name="brand" att_type="string"/>
        <attribute att_name="model" att_type="string"/>
        <attribute att_name="convertible" att_type="boolean"/>
        <attribute att_name="color" att_type="string"/>
        <attribute att_name="mileage" att_type="integer"/>
        <attribute att_name="nr_of_doors" att_type="integer"/>
        <attribute att_name="priceOfCar" att_type="integer"/>
      </business_object>
    </service>
    <service name="provide_contact_information" from="http://www.cardealer.nl/contact/">
    <!-- Provide contact information part-->
    <provides_service_operation>
      <operation name="provide_contact_information">
        <input_parameter type="string">customer_contacts</input_parameter>
        <output_parameter type="void"/>
      </operation>
    </provides_service_operation>
    </service>
    <!-- Provide payment information-->
    <service name="provide_payment_information" from="http://www.cardealer.nl/payment/">
      <provides_service_operation>
        <operation name="provide_payment_information">
          <input_parameter type="string">payment_information</input_parameter>
          <output_parameter type="boolean">confirm|reject</output_parameter>
        </operation>
      </provides_service_operation>
    </service>
```

Fig. 2. An Offerring Service

```
    <!-- Provide shipment information-->
    <!-- Provide shipment information-->
    <service name="provide_shipment_information" from="http://www.cardealer.nl/shipment">
    </service>
  </service>
</SDL_definition>
```

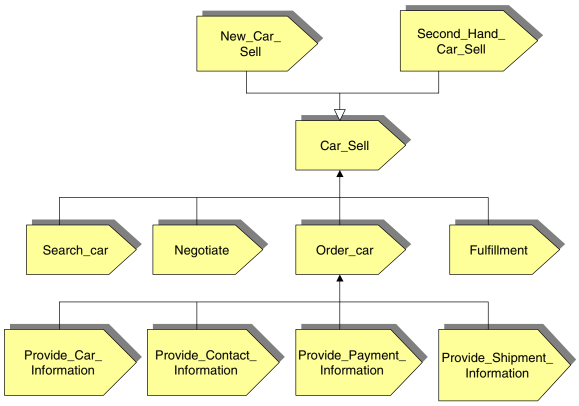


Fig. 3. The e-service model

last categories of information. Like in UDDI, we thus have defined various views on an SDL document: yellow page view, the white page view and the technical, green page view.

5 Service Model

As illustrated in the previous section, an offering service can be recursively decomposed into sub-services or part-services. These sub/part-services represent the capabilities of the offering service. The service hierarchy uses an object-oriented convention in which aggregation and generation are supported so that the sub-services can capture more structural and behavioral semantics of the parent services including the component services. The service hierarchy of the `car seller` in Figure 2 is illustrated in Figure 3.

The combination of inheritance links and part links for the service forms a unified network which improves the reasoning of capabilities and thus maximizes the chance of finding capable service(s) for the request. The algorithm for deriving capabilities for a service is illustrated in Figure 4.

In this paper we derive the capability of each service as a set of services which are either parts of the service, the super services, and their parts as well. The algorithm in Figure 4 starts with the service itself and its part services. Subsequently it continues with the service's super services whose properties are

```
/* recursively compute the capabilities for service $s$ along the hierarchy */
begin
    capability($s$) := $s$;
    for each p in part(s);
        capability($s$) :=capability($s$) $\cup$ $p$;
    endfor;
    for each q in super(s)
        capability($s$) := capability($s$) $\cup$ $q$;
        $s := q$;
        for each p in part(s);
            capability($s$) :=capability($s$) $\cup$ $p$;
        endfor;
    endfor;
end;
```

Fig. 4. Service Capability Derivation Algorithm

```
begin
    Q: the set of request services; FC: Cost function;
    Found:=Null; Cost:=Max_cost;
    For each service $s$ in the registry
        if capability($s$) $\supseteq Q$ then
            $C := FC(s)$;
            if $c < Cost$ then Found:=s; Cost:=C; end;
        end; end;
```

Fig. 5. Matching Requested Service with Offering Services

inherited by the service, and the part service of the super services. The result of this algorithm is a collection of the capabilities of the service. For example, `capability(Order_car)` will be {`Order_car, Provide_car_Information, Provide_Contact_Information, Provide_Payment_Information, Provide_shipment_Information, Car_Sell`}

In our simplified running example we assume that problems such as conflicting naming conventions and semantic mismatches between the requested service (capabilities) and the services in the service repository have been resolved, and the standard vocabularies for service names have been used and registered in the marketplace. Therefore semantically relevant services, i.e., the candidate services for further capability analysis, can be determined by comparing the `vocabulary` lists of the requested service and offerring services in their XML documents (see Figure 1 and Figure 2). In the next section, we shall select the "capable" services among the relevant services based on the capability analysis.

6 Capability Analysis

We categorize the following relationships between the requested service and registered service in terms of their capabilities derived from their description based on the assumption that these services are semantically related:

1. Identical: this means the providing service is the exact match for the requested service as far as the capability concerned. In this case,
$capability(requestService) = capability(provideService)$
2. Part-of: this means the registered service can be used for the requested service partially. In this case,
$capability(requestService) \supset capability(provideService)$
3. More restrictive: this means the registered service can be used by the requested service with some limitation. In this case,
$capability(requestService) = capability(provideService)$, but the results are restricted.
4. More general: this case is the opposite to the above, which means the registered service can be used by the requested service with some constraints.
5. Overlapping: this means only the overlapping part of the functions from the registered service can be used by the requested to achieve required functionality partially. In this case,
$capability(requestService) \cap capability(provideService) \neq \emptyset$
6. Not relevant: this means even they are semantically related, the functions of the registered service can not be used by the requested service. In this case,
$capability(requestService) \cap capability(provideService) = \emptyset$

Using the capability derivation algorithm developed in the previous section, we can discover the above relationships by comparing the capabilities of the requested service with the offerring service. If there is more than one capable service, we must decide on a *matching policy* to select the best service(s). Although there are quite a few factors we need to consider such as cost, quality, workload, etc, here we only take cost into account for illustrative purposes.

The basic algorithm for match making is illustrated in Figure 5. This algorithm compares the capabilities of the offerring services with those of requested service. If the former covers what it is requested, and the cost of the service is within a reasonable price, then the service is selected. It is possible that sometimes we may not be able to find any serivce which can satisfy all the task requirements. Furthermore, certain types of requested service by their nature require multiple services for the cooperative work. To this end, we have also developed a partial match algorithm as shown in Figure 6. This algorithm illustrates that if the capabilities of an offerring service only cover some of the requested capabilities, then the search needs to be continued until the requested capabilities are addressed completely by the set of offerring services. The returning value PS is the set of offerring services which need to be combined to accomplish the requested service.

```
begin
  Q: requested service; PS: partial matching service set;
  PS:={ }; restService:=Q;
  For each $s$ in the registry ordered based on the cost
      if $\exists c \in restService$ and $c \notin capability(s)$ then
         if capability($s$) $\cap$ $restService \neq \emptyset$ then
            $PS := PS \cup s$;
            restService:=restService-capability(s);
         endif;
      endif;
  endfor;
  return PS;
end
```

Fig. 6. Partial Matching Algorithm

7 Service Composition

To gain maximum added-value from e-services, they need to be synthesized into e-service assemblies to support collaborative business process. In addition, e-services should provide mechanisms to dynamically adapt their behavior in terms of provided (call interface) and required (use interface) services promoting dynamic binding and loose coupling, e.g, a worldwide weather information e-service is composed out of hundreds of local weather e-services.

The functionaly that an e-service requires from another e-service is specified in the *use* interface. The functionality or data offered by an e-service and how it behaves when its methods are invoked, is declared in the *call interface*. For example, the order e-service depends on data from the `negotiate` and `search_car` e-service to implement its functions that are laid down in the call e-service. Pluggability requires that the green page information of the e-service, that contains the business object descriptions, is needed to determine the conformance of the actions of the e-services that are about to be integrated. In particular, e-service composition assumes that the call-interface of one component, is conformant to the use-interface of another component.

The problem that needs to be addressed can be described as follows:

Given an use interface description in SDL S_0 and a repository of several SDL-descriptions $S_1 \cdots S_n$, we want to know, which SDL call interface(s) description in the repository fits best to the given SDL description.

Our solution to this problem is to translate an SDL description to a *type-tree*. In a type-tree, a leaf represents a type, a node represents a composite type (such as a method or a state-transition rule). Given two type-trees t_1, t_2 we want to compute a number which is a measurement for the similarity of t_1 and t_2. The higher this similarity coeficient, the greater the similarity.

The similarity of a use interface S_0 to a call interface S_i is a measure for the ability of the requires interface S_i to fulfill the method invocations in the use interface S_0". Note that this measure is not symmetric: possibly is similarity$(S_0, S_i) \neq$ similarity(S_i, S_0). For example, imagine S_0 has three methods and among four methods of S_i are three which perfectly match the three methods of S_0. Neglecting the attributes of S_0 and S_i the component S_i can be used for S_0. But not vice versa: Since S_0, only having three methods can never be a perfect candidate for a four-method-S_i.

This solution encompasses two main phases:

- **Static Type Matching.**
  The first step we envision is to derive a type tree from the SDL specification. The leaves of the tree can only comprise *base* classes, so we need to decompose compound data types down to their base types. Thereafter, the similarity of both type trees is calculated.
- **Dynamic Type Matching / Orchestrating E-services.**
  During this phase, the dynamic information of an e-service specification is checked against other e-service specifications. The dynamic information of components includes: Event-Condition-Action rules (e.g., state transaction rules), state space descriptions as well as the pre- and postconditions of operations.

The static type matching procedure can be performed on the basis of an algorithm that compares two (or more) type trees. The dynamic match between two components can be distilled by looking for similarities between their state machines. The dynamic matching falls outside the scope of this paper. We have worked this out in [9], [11].

In figure 7 and 8, we have depicted a graphical type trees of the relevant SDL specifications for the abovementioned e-services "search_car" and "provide_car_information", that is a part of the composite e-service "order_car".

We now assume that a service interface consists out of service attributes and methods, ignoring the more advanced service concepts that were presented in section 3.2. A measure for similarity(S_0, S_i) in the above sense is:

$$\frac{\mathrm{mapSimple}(\mathrm{Attr}_{S_0}, \mathrm{Attr}_{S_i}) + \mathrm{mapMethods}(\mathrm{Methods}_{S_0}, \mathrm{Methods}_{S_i})}{|\mathrm{Attr}_{S_0}| + |\mathrm{Methods}_{S_0}|} \qquad (1)$$

This definition is the number of mappable items (attributes and methods) divided by the total number of items to map. We use the term mapSimple for the function, which returns the number of mappable attributes because we can reuse its mechanism when mapping function arguments to function arguments.

To ensure type safety on method arguments and method results we require the use of *argument contravariance* (expansion) and *result covariance* (restriction). Method results are said to covariant – they vary in the same way as function types. Result types must be more specific for the function type to be more specific. Conversely, argument types are said to contravariant - they vary in the opposite way as the function type.

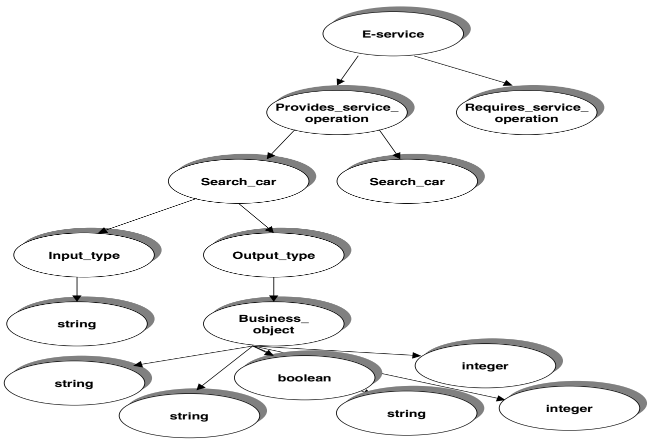


Fig. 7. The Type Tree for the e-service "search_car"

7.1 Composite Type Matching

Let mapMethods(c_i, d_j) denote the similarity of the methods c_i and d_j as respectively specified in the use-interface of an e-service (S_0) and call-interface of two or more e-services (S_i). When mapping method c_i to method d_j, we need to check:

1. whether there exists another $c_h, h \neq i$ with

$$\text{mapMethods}(c_h, d_j) > \text{mapMethods}(c_i, d_j),$$

   and,
2. whether there exists another $d_k, k \neq j$ with

$$\text{mapMethods}(c_i, d_k) > \text{mapMethods}(c_i, d_j)$$

```
mapMethods($\text{Methods}_{S_0}$, $\text{Methods}_{S_i}$)
⟨ fill matrix M⟩
for $c_i$ in $\text{Methods}_{S_0}$ do
    for $d_j$ in $\text{Methods}_{S_i}$ do
        $m_{ij} \leftarrow \text{Methods}(c_i, d_j)$;
    od
od
result _Maxima_Sum (M);
```

The function `mapMethods` builds a Matrix M. Each row r_i of M contains the similarities for a certain method c_i with all provided methods of Methods_{S_i}. The remaining problem is now, to compute an optimal mapping from methods

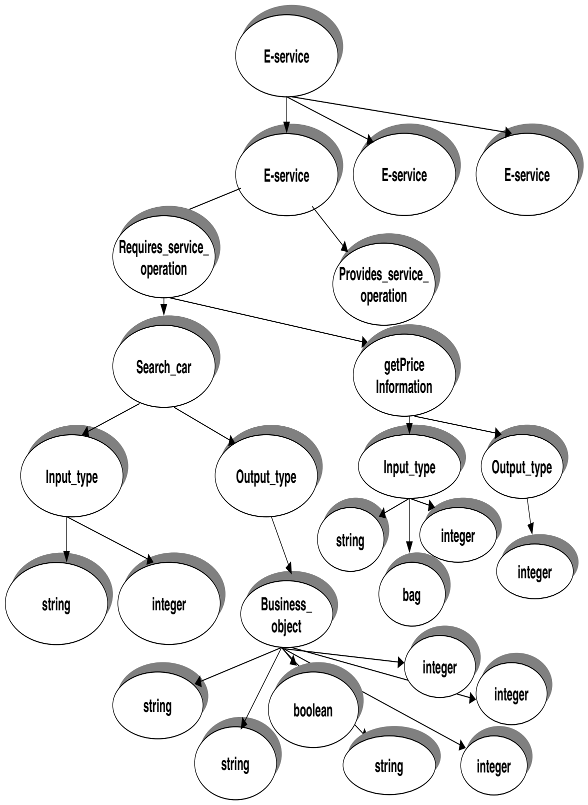


Fig. 8. The Type Tree for e-service "provide_car_information"

of S_0 methods of S_i. This is done by the function `Matrix_Maxima_Sum`. We define the *maxima sum of a matrix* as the maximum sum $\sum_{i \in \{1 \cdots l\}} m_{i\pi(i)}$. That is, we look for a permutation π that makes the above sum maximal. In other words, we select from each row a number, so that the sum of these selected numbers becomes the maximum, but we are not allowed, to select more than one number from each column. Let define the functions $\max_1$ and $\max_2$ as functions which return the highest and the second highest number in a given set of numbers.

With this definition, we can give an algorithm for `Matrix_Maxima_Sum`.

```
**for** $r_i$ of M **do**
    $k_i \leftarrow \max_1(r_i) - \max_2(r_i)$;
**od**
select row $r_i$ with the largest $k_i$; $\langle$ *in case there are more than one (so all with the same $k_i$) select row $r_i$ with the largest $max_1(r_i)$ in case there are several, select indeterministically one of them* $\rangle$
result + $\max_1(r_i)$;
```

```
delete row $r_i$ in $M$;
delete column of $M$ in which $\max_1(i)$ is;

result + MatrixMaximaSum(M);
```

This function returns an optimal mapping because it minimizes the losses by selecting the maxima of each row. A formal proof of this algorithm can be found in [9].

In the running example we have to map the search_car method variants that are provided by the e-service "search_car", to the two methods which are specified in the use-interface of the e-service "order_car".

Our Matrix_Maxima_Sum-Algorithm computes a mapping, with the first function of the search_car mapped to the first of the provides_car_information and the second of the search_car (with a bag as a differentiating input parameter type) to the second method of the provides_car interface (getPriceInformation), as shown in

$$\begin{pmatrix} \mathbf{7/9} & 6/9 \\ 3/4 & \mathbf{3/4} \end{pmatrix} \tag{2}$$

Altogether, Similarity (provides interface(search_car), requires interface (provide_car_information)) $= \frac{7/9+3/4}{2} = \frac{55}{72} (== 76\%)$.

7.2 Interpretation of the Matching Measurement

The measurement that results from our comparison mechanism, expresses the match between two e-service specifications dealing with co- and contravariance. The resulting similarity coefficient now needs to be interpreted by a human analyst, that can the follow various courses of action, depending on the degree of overlap.

The analyst can in principle choose one of the following strategies:

- **Construct a new e-service.**
  If the similarity coefficient is considered to be relatively low, the designer is forced to design and implement his own version of the e-service, and change the use-interface as well as the internal implementation, in such a way that this service can deal with the limited e-services capabilities provided by other existing e-services, and still offer all functionality for which the e-service was designed in the first place.
- **Change the call-interface.**
  Change the call-interface of the service under development which may result in also changing the use-interface of this e-service. This is assumed to be done in such a way that the use-interface of the e-service under development is conformant with the call-interfaces of required e-services.

The scenarios presented need to be worked out on the basis of various field experiments we intend to conduct in the near future.

8 Conclusions

In this paper we have discussed the representation, discovery and composition of e-services for e-market places. A framework was introduced for describing services in terms of a declarative language and for creating and composing new services. We also introduced simple algorithms for service matching in terms of service capabilities. Service capabilities describe services that can be part of a service, an e-services super-services and their constituent part services. Finally, we have discussed a technique for describing service composition by translating service specifications in SDL to type trees and an algorithm for comparing the similarity of these type trees.

References

1. F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, M. C. Shan. *Adaptive and Dynamic Service Composition in eFlow*, HP Lab. Techn. Report, HPL-2000-39. 272
2. S. Chen. *Retrieval of Reusable Components in a Deductive, Object-Oriented Environment*. PhD thesis, RWTH Aachen, Information Systems Institute, 1993.
3. D. Georgakopoulos, H. Schuster, D. Baker, and A. Cichocki. *Managing Escalation of Collaboration Processes in Crisis Mitigation Situations*, Proceedings of ICDE 2000, San Diego, CA, USA, 2000. 272
4. Scott Henninger. *An Evolutionary Approach to Constructing Effective Software Reuse Repositories*, ACM Transactions on Software Engineering and Methodology", vol. 6, nr. 2, pp. 111–140, 1997.
5. International Business Machines (IBM). *Energize e-business with Web services from the IBM Websphere software platform*, http://www-106.ibm.com/developerworks/library/ibm-lunar.html IBM, 2000. 271
6. International Business Machines (IBM). *The Web Service Description Language* $http : //www - 106.ibm.com/developerworks/library/w - wsdl.html$ IBM, 2000 271
7. J.-J. Jeng and B. H. C. Cheng. *Specification Matching for Software Reuse: A Foundation*, Proceedings of the the 17th international conference on software engineering on Symposium on software reusability, 1995, Pages 97 - 105
8. G. Joeris and O. Herzog. *Managing Evolving Workflow Specifications with Schema Versioning and Migration Rules*, TZI Technical Report 15, University of Bremen, 1999. 272
9. W. J. van den Heuvel and R. Reussner. *Matching Component Interfaces*, Technical Report, Universität Karlsruhe 2001. 280, 283
10. UDDI.org *UDDI Technical White paper*, $http : //www.uddi.org/pubs/lru\_UDDI\_Technical\_Paper.pdf$, 2001 271
11. R. Reussner. Formal Foundations of Dynamic Types for Software Components. Technical Report 08/2000, Universität Karlsruhe, Department of Informatics, 2000 280

Schema Design and Query Processing in a Federated Multimedia Database System

Henrike Berthold and Klaus Meyer-Wegener

Dresden University of Technology, Computer Science Department,
01062 Dresden, Germany
{henrike.berthold,kmw}@inf.tu-dresden.de

Abstract. To build information systems today, a multimedia database system is needed which supports both structured data and media data. Because these two kinds of data differ significantly in their characteristics, the extension of traditional databases towards the support of media data is a very tedious task. As an alternative, the intelligent coupling of specialized component systems, i.e. traditional databases, media storage systems, and media retrieval systems, can be used to build a multimedia database system. In this paper, a global-schema construction methodology for a federation of these component systems is presented. It allows the unrestricted extension of the global schema. Furthermore, a schema-independent query processing module for global schemas constructed with this methodology is designed. It can handle the given distribution pattern. In addition, it explores the functionality of media retrieval systems concerning search functions and result restrictions.

1 Introduction and Motivation

In today's information systems, a multimedia database system is needed which supports both structured data and media data. Because these two kinds of data differ significantly in their characteristics, the extension of traditional databases towards the support of media data is a very tedious task. However, a lot of work has already been dedicated to the development of specialized systems for either media storage or media retrieval. Hence, an intelligent coupling of these systems with traditional databases can provide the combined functionality.

A major achievement of database systems is the support of a powerful query language which is used on a schema to extract information in a flexible way. Hence, a global schema and a query language should also exist for the coupling. Furthermore, it must be considered that the component systems are heterogeneous and independently developed. In research, different degrees of database coupling have been proposed. A tightly coupled federated database system [20] solves the coupling problem. It has a global schema, while it still allows its component systems some degree of autonomy. Hence, we propose a tightly-coupled "Federated Multimedia Database System" (FMDBS).

In this paper a methodology to construct the global schema based on existing source schemas and search functions is presented. Furthermore, the FMDBS-

© Springer-Verlag Berlin Heidelberg 2001

specific part of the query processing which is related to the global schema construction is shown.

This paper is structured as follows. The basic architecture of a Federated Multimedia Database System and its potential component systems are described in Sect. 2. The global schema design methodology is presented in Sect. 3. The query processor based on this methodology and its implementation are described in Sect. 4. The closing Sect. 5 summarizes and concludes the paper.

2 A Federated Multimedia Database System

An FMDBS federates the following kinds of component systems:

- traditional database systems managing structured data (abbr.: SDBS),
- media storage systems (abbr.: MSS), and
- media retrieval systems doing content-based search (abbr.: MRS)

via a federation management system (FMDBMS) (see Fig. 1). Each component system has an associated wrapper which adapts the proprietary interface and data model to the needs of the FMDBS.

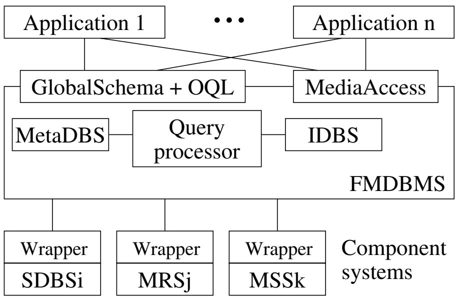


Fig. 1. Part of the FMDBS architecture

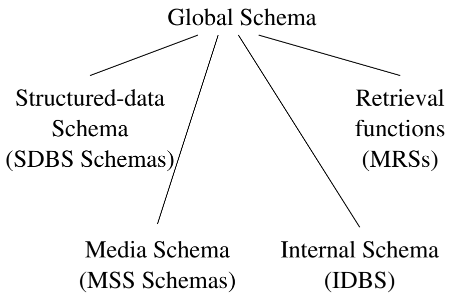

Global Schema
Structured-data Schema (SDBS Schemas)
Retrieval functions (MRSs)
Media Schema (MSS Schemas)
Internal Schema (IDBS)

Fig. 2. FMDBS Schema architecture

The federation management system includes the query processor which handles global queries. It uses the Meta Database System (MetaDBS) and the Internal Database System (IDBS) (see sections 3 and 4). Details of the FMDMBS regarding transaction processing and the specifics of media access operations are discussed elsewhere [2].

An FMDBS global schema only includes media descriptions (data about the media objects such as size and format). It does not comprise media access operations because they cannot be used in the collection-oriented access offered by a database query language. Media description and media access operations are connected with the help of a Media Access Identifier (MID). The MID can be retrieved via an OQL query. It is then used as a parameter of the media access operations.

In the following sections, existing media storage systems (Sect. 2.1) and media retrieval systems (Sect. 2.2) are evaluated for their use in the FMDBS. Since database systems for structured data are well established, they are not included in the overview. Finally, in Sect. 2.3, related work in terms of systems and ongoing projects which aim to manage structured and media data is described.

2.1 Media Storage Systems

To be useful here, media storage systems must provide appropriate interfaces for external control. Furthermore, they must be focused on the storage and management of single-media objects. In that, they must support especially timed media data like video, which are large-sized and need to be streamed with certain timing constraints. They may also provide media descriptions for their objects and collection management, which will be used in the construction of the global schema. Fellini [16] and the Tiger Video Fileserver [3] represent all data uniformly as files. Media descriptions such as format can only be derived from the file name or file characteristics. AMOS Audio [14] and Kangaroo [15] provide explicit information about stored media data such as resolution and format. The file system Fellini organizes data in directories. Kangaroo media objects can be grouped in named sets. The Tiger Video Filesystem does not provide collection management, i.e. data are accessed by name or address.

2.2 Media Retrieval Systems

Media retrieval systems facilitate content-based retrieval using features automatically extracted from the raw media data. Most of them are specialized to one media type. An FMDBS can only use open systems, but most retrieval systems have just an interactive interface, and no programming interface. The exceptions are QBIC [9] and the Excalibur Image DataBlade [13]. The latter can be used via a database language binding.

To integrate the functionality of such systems into the FMDBS global schema, their search functions must be studied. The typical search argument is an example media object. Alternatively, features can be specified directly such as a color distribution. Furthermore, the features may be weighted.

There are two basic search functions. The first determines a value for the similarity between a media object and the search argument. This function is called single. Such a function exists e.g. in the QBIC API and the Excalibur Image DataBlade. Since media retrieval systems are attached to a collection of media objects, they provide a second function which calculates the similarity of *all* media objects in the collection with the provided search argument. The function's result is typically a list of tuples consisting of a media object identifier and the corresponding similarity value, ordered by descending similarity value. This function is called coll. At the first glance, it seems to be just a repeated invocation of the function single, but it is implemented differently, e.g. using specific access paths. Therefore, it is more efficient.

Some systems allow to further restrict the result list of the function coll. Specifying such restrictions can improve the performance, especially for large media collections. For instance, the QBIC API and the Excalibur Image DataBlade allow to specify a lower bound for the similarity. QBIC additionally supports the specification of a subset of media objects to be searched, and the specification of maximum size of the result list (i.e. “top n”).

2.3 Related Work

This section presents work that also has the goal to build a multimedia database system, i.e. to manage both structured data and media data in a single database system. There are two approaches: One couples component systems for both types of data (coupling approach), and the other extends a traditional database system towards the support of media data (extension approach).

HERMES [23], TSIMMIS [6], and Garlic [4] couple heterogeneous data sources such as databases, spatial data, text data, and images. HERMES and TSIMMIS do not handle media data in a specific way, and both systems do not provide a global schema. The Garlic system provides a global object-oriented schema, which can be queried and manipulated using an object-oriented SQL-like language. It supports media search systems in a specific way. This has been demonstrated by the integration of the image retrieval system QBIC [7]. Each Garlic data source is represented by its own set of object types and classes in the global schema. Hence, the simultaneous use of two media data sources, e.g. a video server and a video retrieval system, raises the problem of duplicate classes. Furthermore, objects in different sources can only be combined by using “complex objects” (stored in the Garlic Complex Object Repository). Most of these combinations could instead be achieved through object relationships, which would lead to a simpler and clearer schema.

For the extension approach, today’s ORDBMS can be used with packages that define data types and functions [22]. Media packages are the Excalibur Image DataBlade (EIDB) [13], the Informix Video Foundation DataBlade (VFDB) [12], and the Continuous Long Field DataBlade (CLFDB) [11]. The new standard SQL/MM [21] specifies media extensions based on the standard SQL:1999. The parts for text and spatial data passed the final standardization step recently. The image extension proposal is currently in the standardization process. Up to now, there are no parts for audio and video. Small-sized media data like images can be stored inside a database. Large-sized continuous media data today can only be stored externally, because database managers do not support streaming operations. This can lead to consistency problems when operations on media data are performed in the external media server without the database knowing them. Furthermore, simultaneously using independently developed extensions such as a video storage and a video retrieval extension is complicated, because each extension manages its own data.

3 Integration Methodology

The construction of the global schema from existing component schemas is called integration. The integration must be performed by an expert who knows the component systems and applications. To ease the query processing task, two intermediate schemas are introduced: the structured-data schema and the media schema (see Fig. 2). The structured-data schema represents the data of all database components, i.e. it describes a virtual database system for structured data (SDDBS). It is constructed using one of the well-known integration strategies; an overview of such techniques is given in [18]. The media schema represents all media descriptions, i.e. it describes a virtual media database system (MDBS). Its construction is described in Sect. 3.1. The integration process of the two intermediate schemas and the media search functions (Sect. 2.2) comprises three steps. Each of them is informally described in the following sections. The formal version is given in [2].

The data model and the query language of the global schema are those proposed in the CROQUE project [19]. The goal of this project was to describe formally the ODMG data model and query language [5]. An CROQUE database schema consists of classes and types. A type describes the characteristics of an object, i.e. attributes, relationships and methods. A class is a collection of objects with a type assigned. Figure 3 shows a combined class and type hierarchy. The class name is written before the colon, and the type associated with this class is given behind the colon. The is-a relations in this hierarchy are based on the definition of a subclass. A subclass holds a subset of the objects of its superclass. The type associated with the subclass is equal to or a subtype of the type associated with its superclass. Hence, the hierarchy also shows the subtype relations.

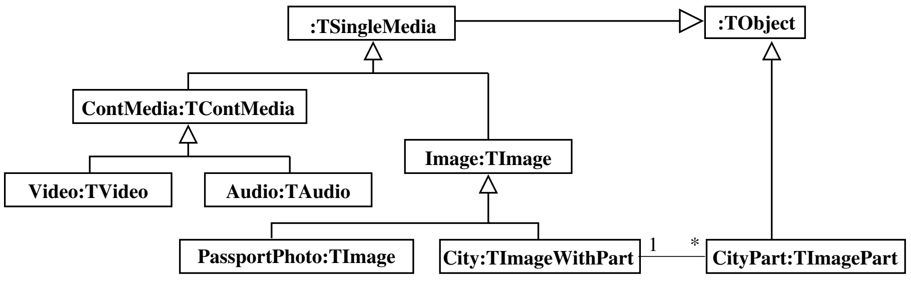


Fig. 3. Example media schema

3.1 Media Schema

Each media storage component manages media objects which are uniquely identified by an MID. The media description which is associated with a media object must be represented by one or more objects in the media schema. The number of these objects is a matter of modeling. A media description is modeled by a proxy media object (i.e. an object with an MID) and zero or more description objects. Each media schema includes the types TObject and TSingleMedia (see example media schema in Fig. 3). TObject is the base type for all objects. It defines the OID attribute. The type TSingleMedia represents all proxy media objects. It extends type TObject by the MID attribute.

The media description is different for each media type. For example, a video has other characteristics than a text. Therefore, the media schema has exactly one proxy media type for each media type which is managed by one of the media storage components (TAudio, TVideo, and TImage in Fig. 3). The set of characteristics is inalterable in a media storage system, i.e. it cannot be changed by an application or a user. Ideally, a uniform representation of the media descriptions (i.e. a common set of characteristics' names and types) of a media type can be found for all media storage components (cf. TAudio and TVideo in Fig. 3). Semantical and syntactical heterogeneities e.g. different names for the same thing ("frame_rate" vs. "frames_per_second") must be identified. The mapping to the chosen representation must be performed by the wrappers. However, if a uniform representation cannot be found for a media type, a type hierarchy with one base proxy media type and subtypes for different representations must be used (TImage in Fig. 3). This type hierarchy is needed to allow for classes which are independent of the objects' sources. The subtype TImageWithParts of TImage in Figure 3 is extended by a relationship with sets of objects describing image parts. The type TImagePart is an example of a pure description type. Furthermore, types with common characteristics of media types can be defined in the media schema, e.g. TContMedia. A common attribute for all continuous media objects is e.g. the duration.

The semantics of media objects are handled by the MSSs and are exported as operations of the media data types. The design of those types (primarily the selection and specification of the operations) is still an open issue. The proposals in existing systems and in the standardization process (SQL/MM [21]) are not considered sufficient, since they lack data independence and powerful access operation. Consequently, FMDBS must be open for future improvements in the definition of media data types, particularly with respect to their semantics as expressed in the operations.

Classes should be defined for querying and the association with media retrieval systems. Each media storage component manages collections of media objects. The wrapper maps these collections onto the concept of the media storage system, e.g. to directories in file systems.

Since the grouping to collections is not constrained in most media storage systems, for example in file systems, the following requirements must be fulfilled by the media storage components to allow media schema construction. First, each media storage component must provide a collection for each media type it manages ("basic media collection"). A basic media collection must contain all media objects of the corresponding media type in the media storage system. Second, a subcollection hierarchy, which is based on the subset relation, can exist below the basic media collections. Finally, it must be possible to add subcollections.

In the media schema, there is one class for each media type managed by one of the media storage components ("basic media classes"; Audio, Video, and Image in Fig. 3). These classes unite the basic media collections of the media storage systems.

Sub- and subsubcollections of the basic media collections are represented as sub- and subsubclasses in the media schema. The semantics of subcollections defined in different media storage systems determines whether or not they can be merged into one class. Most of these classes (e.g. PassportPhoto in Fig. 3) are associated with the corresponding base proxy media types. To each of these classes a corresponding collection must be defined in the media storage components which do not have an appropriate collection but support this media type. This has the advantage that the addition of an object to another class, which is associated with the same type, does not require to create the media object in another media storage system. The classes in the media schema are unions of these collections. For some classes of media objects, special information which is provided as part of the media description only by few media storage components, should be available (e.g. class City in Fig. 3). These classes have a subtype of the base proxy media type. Corresponding collections must be defined in all media storage components which provide this information.

Furthermore, depending on the applications' needs, the media schema may hold classes which unite basic media classes or classes of different media types (see class ContMedia in Fig. 3).

3.2 Base Integration and Addition of Media Search Functions

In the first step, the structured-data schema and the media schema are integrated. Their name spaces and contents are disjunct. Hence, the base integration is done simply by creating a global base type.

In the second step, media search functions are integrated into the global schema. This enables queries that combine boolean search for structured data with content-based search for media objects.

A media retrieval system provides search functions for a collection of media objects, i.e. a class of proxy media objects in the global schema. A method in the type of this class is perfectly suited for the single function. Hence, the global type must be extended by a corresponding method. If the type is associated with multiple classes, a new global type which subtypes the original one must be added and must then be associated with this class only. The coll function should be turned into a class method, because it evaluates all objects in the class. However, user-defined class characteristics do not exist in ODMG ODL. They are not implemented in most existing DBMS, either. Hence, coll cannot be included in the global schema. However, for single, the query processor can decide to use coll internally when single is used in an appropriate query (see Sect. 4 for details).

3.3 Schema Extension

Finally, in the third step, the global schema is extended. The two schema parts in the global schema are independent so far, but in most cases structured-data objects and media objects are related. The insertion of a relationship is called a *simple extension*. It can have two basic meanings. First, media objects may represent real-world objects, e.g. persons or events, that are modeled as structured-data objects. Second, structured-data objects may hold information about media objects, e.g. producer or date of production. A simple extension requires the extension of the types of the two related classes.

The addition of a characteristic f (attribute, method, or relationship) to a class c with type t is performed as follows. A new type t' that extends t by f is created. For each subtype u of t a subtype u' of t' is created that is also extended by f. The type graph between u' and t' is copied from the type graph between u and t. Only the nodes (i.e. types) are changed to the corresponding extended types. Class c is then assigned to type t'. Each subclass d of c is associated with the corresponding extended type. Type t' becomes a new subtype of t. Furthermore, each extended subtype u' becomes a subtype of its original type u. This procedure is shown in Fig. 4 for class $c1$ with type $t1$.

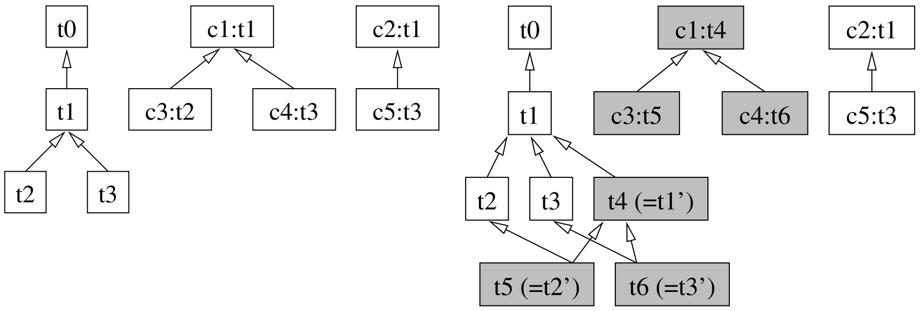


Fig. 4. Extension of global class $c1$ with type $t1$ (original/extended global schema)

The global class c has a corresponding class e with type v in the IDBS. A new type v' that extends v by f is created. The internal class and type hierarchy is extended in the same way as the global class and type hierarchy.

Adding relationships between media and structured-data objects is not sufficient to express more complex things, e.g. the combination of single media objects to form multimedia objects. Such an extension is called *complex extension*. It requires the addition of new types and classes and the extension of existing types. New types are added to the global schema and the internal schema below the base type. Classes of these types are added beside existing classes. An example schema for multimedia lectures is shown in Fig. 5. Lectures and slides are defined in the structured-data schema (light gray). Video, text, and image objects are defined in the media schema (dark gray). A video is the recording of a lecture (simple extension). An image shows a slide, and a text describes it (simple extensions). The media objects must be arranged in space and time. This arrangement is described by the new type TVisualElement. Its subtypes are extended with a relationship to the corresponding proxy media object (complex extensions).

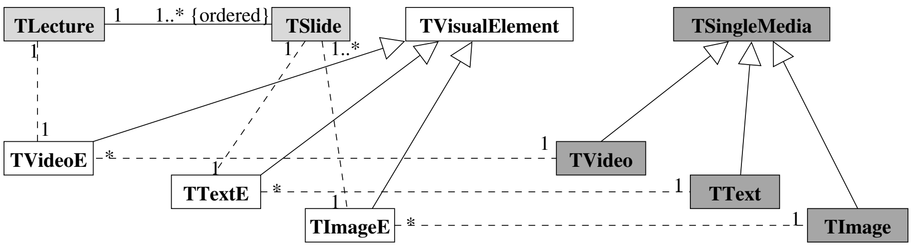


Fig. 5. Model for multimedia lectures

The objects of the new types and extensions are stored in the internal database (IDBS, see Fig. 1). The latter are stored as "difference objects" which have an OID and the new characteristics. The schema of the internal database has the same class structure as the global database. Each object is at least represented by its OID. The type structure is determined by the extensions.

3.4 Construction Data

The data which describe the construction of the global objects from source objects are called "construction data". They are stored in the MetaDBS (Fig. 1). The query processor uses them and the description of each participating schema to execute global queries. The construction data comprise the following elements:

– Each global class has exactly one source class in the IDBS. It may have an additional source class, either in the MDBS or in the SDDBS. Both are retrieved via the function *SCoGC (Source Class of Global Class).*
– Each global type has exactly one source type in the IDBS. It may have an additional source type, either in the SDDBS or in the MDBS. The function *SToGT (Source Type of Global Type)* retrieves them.
– Each characteristic in a global type has exactly one source. A source is either a database component, i.e. SDDBS, MDBS, or IDBS, or a media retrieval component. They are retrieved by *SoGC (Source of Global Characteristic).*

Furthermore, several propositions hold for the assembly of the global schema. They are enforced by the integration methodology. They are used to prove the correctness of the query processing.

P1 Each global class has an assigned source class in the IDBS. The set of objects in the global class is equal to the sets of objects in the assigned internal class.
P2 The set of global objects having a global type is equal to or a subset of the set of source objects having the source type that is assigned to the global type.
P3 A global type with a characteristic which is defined in a source has an associated type with this characteristic in this source.
P4 A global type to which a media retrieval function is applied has an assigned type in the MDBS which has the characteristic mid.

4 Query Processing

Queries in federated databases are processed in multiple steps [17]. In this section, each of these steps is explained. Specific aspects of FMDBMS query processing are emphasized.

The first step, which is called "global query computation", includes the check whether the query is correct, and the mapping to the intermediate representation. In the second step, the global query is modified. The specifiers of the global schema, e.g. class names, are replaced by source specifiers using the construction data. This replacement depends on the distribution pattern. A typical approach is to replace a global specifier that represents a collection of data, i.e. a table or a class, by a query that constructs this collection using local specifiers only. It does not work, however, for the FMDBMS distribution pattern, in which types are vertically fragmented because some parts of the objects are stored in the media database or the structured database and other parts are stored in the internal database. Hence, a relationship in the global schema refers to a type other than the associated relationship in the data source. The complete replacement of global objects would require to construct all objects which can be reached via a global path (that is a sequence of relationships). Hence, in a query that would replace a collection of objects of a vertical fragmented type, each relationship that refers to a vertically fragmented global type again must be represented by subquery. This must be repeated recursively until an unfragmented type or a fragmented type without relationships is reached. This implies that another replacement scheme must be applied. Furthermore, in the FMDBMS special attention must be paid to the use of media retrieval components. Then, in the third step, a query is decomposed into subqueries and postprocessing queries, and it is optimized globally. A subquery is a query that needs only data from a single component system. Postprocessing queries combine the results of the subqueries. In step four, the subqueries are optimized and executed by the component systems. Finally, the global result is built from local results and postprocessing queries.

The significant difference from other federations is the modification due to the distribution pattern and the use of media search functions. A modification algorithm is presented in Sect. 4.2. In Sect. 4.3 it is described how media search functions can be exploited. The prototype implementation is shortly presented in Sect. 4.4. An intermediate representation should ease query manipulation. A well chosen representation reduces the amount of work necessary to do the required transformations. Sect. 4.1 motivates and introduces the representation used for the modification.

4.1 Intermediate Representation

The intermediate representation of an object query language can be either an object algebra or comprehensions based on monoids or monads [8,10]. Object algebras are very complex compared to the relational algebra because the object model is much richer than the relational model. Most object algebras have a set of operators for each collection type (set, bag, ...) and conversion operators. However, comprehensions based on monoids or monads represent object queries in a uniform way. Furthermore, a comprehension expression can be transformed to an algebraic expression [10]. This allows to use comprehensions in combination with an object algebra. Therefore, comprehensions have been chosen as intermediate representation for the modification.

The comprehension syntax [10] enriches a basic expression language like the lambda calculus with comprehensions. In this syntax, there are three collection types T, namely set, bag, and list, and six non-collection types, namely exists, all, sum, prod, max, and min. A comprehension $[\![e \mid\!] q_1, ..., q_n]\!]^T$ consists of a head expression e and qualifiers q_i. A qualifier is either a generator $v \leftarrow q$ that sequentially binds variable v to the elements of range q, or a filter p which is a boolean expression. The variable v of generator q_i is bound in q_{i+1}, ..., q_n and e. The binding of v is propagated until a filter evaluates to false under this binding. The head expression is evaluated for each binding that passes all qualifiers. The result of the head evaluation is added to the comprehension result via a T-specific operation e.g. set-insertion or max-determination.

4.2 Modification

In the modification step, global specifiers are replaced by local specifiers. Global specifiers occur at different places in a query because expressions can be combined in OQL as long as the types are correct. Therefore, a recursive modification function $\mathcal{M}$ is defined. The rule for which the argument matches the actual query representation part is executed. The modification function uses the construction data via functions introduced in Sect. 3.4. To improve readability, the function *OoTaID (Object of Type and OID)* is introduced. It returns an object given its OID and a type.

Since global objects can be vertically fragmented, the modification does not construct objects fully. Instead, objects are represented as base objects which have only the OID attribute. Base objects are constructed via the function *BO* (Base Object).

The modification function has only two rules (rule 1 and 5 shown below) which change the query representation. Rule 1 handles global class names. Rule 5 replaces the application of characteristics. The other rules are necessary to walk through the query representation recursively.

1. Global class name
$\mathcal{M}(n_c^G : Tset(Tobj)) = BO(n_c^I)$
with $n_c^I = find(IDBS, SCoGC(n_c^G))$
The function $find$ returns the name of the class that is associated with the source IDBS in the result of $SCoGC(n_c^G)$.

5. Characteristic application on an object expression
$\mathcal{M}((e : Tobj).app(par))$
Let $s = SoGC(app)$, $par' = \mathcal{M}par$, and $e' = \mathcal{M}e$.
Since each object is only represented by its OID attribute, the characteristic *app* cannot be applied directly. The source s of the function *app* is determined via $SoGC$. The source type *st* is determined using s and the global type $Tobj$. The object having type *st* and the given OID is determined using $OoTaID$. When the source s is a media retrieval component, then the function single of this media retrieval component is used. When the source is a database, then the function *app* is applied to the constructed object. When the result of *app* is an object or a collection of objects, then the corresponding base objects are produced via BO.

1. $s \notin \{MDBS, SDDBS, IDBS\}$ i.e. *app* is a retrieval method
   $\mathcal{M}(e.app(par)) = single_s(oexp.mid, par')$
   with $st = find(MDBS, SToGT(Tobj))$ and $oexp = OoTaID(st, e'.id)$
2. $s \in \{MDBS, SDDBS, IDBS\}$ and result has type
   (a) Tobj, Tcoll Tobj
   $\mathcal{M}(e.app(par)) = BO(oexp.app(par'))$
   with $st = find(s, SToGT(Tobj))$ and $oexp = OoTaID(st, e'.id)$
   (b) Tbase or a type constructed from Tbase
   $\mathcal{M}(e.app(par)) = oexp.app(par')$

During the modification, non-equivalent transformations (in the sense of algebraic equivalences) are executed because additional information is used. The correctness of rule 5.1 is shown in the appendix. The correctness and completeness of the modification algorithm can be found in [2]. The modification can be improved. For the following cases, improvements are described in [2].

– Paths belonging entirely to one source can be replaced as a whole instead of replacing each single relationship within the path.
– A generator which binds a global object can be replaced by a sequence of qualifiers producing all local objects which are used in the query.

4.3 Improved Use of Media Retrieval Components

The modification function replaces each search method by the function single of the corresponding media retrieval component. But in case of repeated calls with the same parameter, the function coll should be used instead. Furthermore, result restrictions for the function coll which are supported by a media retrieval component should be exploited. Such result restrictions can be derived from a query representation using pattern matching. In this section, two rules are presented. Furthermore, the correctness of one rule is shown in the appendix. Further rules and correctness proofs can be found in [2].

– Replace function single by function coll
  A comprehension which conforms to the pattern
  $[\![a[single_s(p.mid, b)] \;[\!]\; ps1, p \leftarrow c, ps2[single_s(p.mid, b)]]\!]^T$
  ($a[b]$ denotes the existence of expression b in expression a)
  and the condition
  (C1) expression b is constant, i.e. it does not contain a free generator variable, is replaced by
  $[\![a[r.sv/single_s(p.mid, b)] \;[\!]\; ps1, p \leftarrow c, r \leftarrow coll_s(b), r.mid = p.mid,$
  $ps2[r.sv/single_s(p.mid, b)]]\!]^T$
  ($a[b/c]$ denotes the replacement of expression c by b in expression a)
– Identify a lower-bound restriction
  A comprehension which conforms to the pattern
  $[\![a \;[\!]\; ps1, r \leftarrow coll_s(par), ps2, r.sv >= b, ps3]\!]^T$
  and the condition (C1) that expression b is constant is replaced by
  $[\![a \;[\!]\; ps1, r \leftarrow coll_s(par, lower\_bound(b)), ps2, ps3]\!]^T$

4.4 Implementation

A prototypical FMDBS query processor has been implemented in SML/NJ0.93. For the type checking and mapping to the comprehension representation, code developed within the CROQUE project has been reused. The modification applies the two general improvements and the replacement of the function single. The final program generation bases on the comprehension-to-expression mapping [10]. It generates C++ programs. These programs use the O2 [1] C++ binding to access the source O2 databases (MDBS, SDDBS and IDBS). A media retrieval system is simulated with a C++ program. Compiling and linking these programs lead to an executable which produces the result. Example schemas have been used to demonstrate the query processing.

5 Summary and Outlook

In this paper, a global-schema construction methodology and a schema-independent query processing module for global schemas constructed using this methodology are shown. The construction methodology allows the free extension of the global schema. The query processing can handle the distribution pattern which results from the application of the construction methodology. Furthermore, it can explore the capabilities of media retrieval systems (function coll and result restrictions). The use of the function coll and the result restrictions improves the efficiency of the media retrieval.

Future work is necessary to include the use of media search functions in the global optimization process. Research questions include:

- Is there a general cost model for media retrieval systems ?
- How to derive costs for unknown media retrieval systems ?
- Which result restrictions are useful to apply for which query ?

Acknowledgements

This work has been supported by the Deutsche Forschungsgemeinschaft (DFG) under the grants GRK 191 and SFB 358. The authors also thank the anonymous reviewers for their useful hints on how to improve the presentation.

A Proof of Correctness of Modification Rule 5.1

It holds:
(1) The source of *app* is a media retrieval component (prerequisite to rule 5.1).
(2) The modification $\mathcal{M}$ is correct.
(3) Function single is defined in each media retrieval component. It returns the similarity value.

Proof:
$2 \Rightarrow e = e'$ (4)
$2 \Rightarrow par = par'$ (5)
1,P4 $\Rightarrow$ There is a source type st to $Tobj$ in the MDBS. (6)
P2,6 $\Rightarrow$ There is an object with type st in the MDBS that has the same OID than the global object.(7)
P4,DEF of OoTaID,4,7 $\Rightarrow$ The object OoTaID(st,e'.id) has the function mid. (8)
3,5,8 $\Rightarrow$ $e.app(par) = single_s(OoTaID(st, e'.id).mid, par')$ q.e.d.

B Proof of Correctness of the Replacement of Function single by Function coll

It holds:
(1) M is the media class to which the media retrieval system s is assigned.
(2) $coll_s(par)$ is defined as $[\![mid : m.mid, sv : single_s(m.mid, par) \parallel m \leftarrow M]\!]^{list}$ with par is constant.
(3) The MID is a unique identifier of a proxy media object.
(4) $c \subseteq M$ (This holds because the query expression is correct.)
(5) Normalization rule which has been proven in [10]: $[\![e1 \parallel qs1, v \leftarrow [\![e2 \parallel qs2]\!]^{T1}, qs3]\!]^{T2} = [\![e1[e2/v] \parallel qs1, qs2, qs3[e2/v]]\!]^{T2}$

Proof:
$[\![a[r.sv] \parallel ps1, p \leftarrow c, r \leftarrow coll_s(b), r.mid = p.mid, ps2[r.sv]]\!]^T$
= (C1),(2)

$[\![a[r.sv] \mid\!] \; ps1, p \leftarrow c, r \leftarrow [\![mid : m.mid, sv : single_s(m.mid, b) \mid\!] \; m \leftarrow M]\!]^{list},$
$r.mid = p.mid, ps2[r.sv]]\!]^T$
$= (5)$
$[\![a[single_s(m.mid, b)/r.sv] \mid\!] \; ps1, p \leftarrow c, m \leftarrow M, m.mid = p.mid,$
$ps2[single_s(m.mid, b)/r.sv]]\!]^T$
$= (3),(4)$
$[\![a[single_s(p.mid, b)] \mid\!] \; ps1, p \leftarrow c, ps2[single_s(p.mid, b)]]\!]^T$ q.e.d.

References

1. F. Bancilhon, C. Delobel, and P. C. Kanellakis, editors. Building an Object-Oriented Database System, The Story of O2. Morgan Kaufmann, 1992. 297
2. H. Berthold. A Federated Multimedia Database System. PhD thesis, Dresden University of Technology, Germany, 2001. In preparation. 286, 289, 296, 297
3. W. J. Bolosky et al. The Tiger Video Fileserver. In Proc. of 6th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 96), 1996. 287
4. M. Carey et al. Towards Heterogenous Multimedia Information Systems: The Garlic Approach. In Proc. of the Fifth International Workshop on Research Issues in Data Engineering(RIDE): Distributed Object Management, 1995. 288
5. R. Cattell et al., editors. The Object Database Standard: ODMG 2.0. Morgan Kaufmann, 1997. 289
6. S. Chawathe et al. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In Proc. of IPSJ Conference, 1994. 288
7. W. F. Cody et al. Querying Multimedia Data from Multiple Repositories by Content: the GARLIC Project. In Proc. of IFIP 2.6 Third Working Conference on Visual Database Systems (VDB-3), 1995. 288
8. L. Fegaras and D. Maier. Towards an Effective Calculus for Object Query Languages. In Proc. of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, May 22-25, 1995, pages 47–58. ACM Press, 1995. 295
9. M. Flickner, H. S. Sawhney, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by Image and Video Content: The QBIC System. IEEE Computer, 28(9):23–32, 1995. 287
10. T. Grust. Comprehending Queries. PhD thesis, Universität Konstanz, 1999. 295, 297, 298
11. S. Hollfelder et al. Transparent Integration of Continuous Media Support into a Multimedia DBMS. In Proc. of Int. Workshop on Issues and Applications of Database Technology (IADT'98), 1998. 288
12. Informix Corp. Informix Video Foundation DataBlade Module Users's Guide, 1998. 288
13. Informix Corp. Excalibur Image DataBlade Module User's Guide, 1999. 287, 288
14. M. Löhr and T. C. Rakow. Audio support for an object-oriented database management system. Multimedia Systems, 3(5,6):286–297, 1995. 287
15. U. Marder and G. Robbert. The KANGAROO Project. In Proc. of the Third Int. Workshop on Multimedia Information Systems (MIS'97), 1997. 287
16. C. Martin, P. Narayan, B. Özden, R. Rastogi, and A. Silberschatz. The Fellini Multimedia Storage System. Journal of Digital Libraries, 1997. http://www.bell-labs.com/project/fellini/papers.html. 287

17. W. Meng and C. Yu. Query Processing in Multidatabase Systems. In W. Kim, editor, Modern database systems : the object model, interoperability, and beyond. Addison Wesley, 1995. 294
18. E. Pitoura, O. A. Bukhres, and A. K. Elmagarmid. Object Orientation in Multidatabase Systems. ACM Computing Surveys, 27(2):141–195, 1995. 289
19. H. Riedel and M. H. Scholl. A Formalization of ODMG Queries. In Proc. of the IFIP WG 2.6 Working Conference on Database Semantics (DS-7), Leysin, Switzerland, 1997. 289
20. A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. ACM TODS, 22(3), 1990. 285
21. K. Stolze. SQL/MM Part 5: Still Image - The Standard and Implementation Aspects. In Proc. of Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), 9. GI-Fachtagung, Oldenburg, 7.-9. März 2001, Informatik Aktuell. Springer, 2001. 288, 290
22. M. Stonebraker and P. Brown. Object-Relational DBMSs Tracking the Next Great Wave. Morgan Kaufmann, 1999. 288
23. V. Subrahmanian, S. Adali, A. Brink, J. J. Lu, A. Rajput, T. J. Rogers, R. Ross, and C. Ward. HERMES: A Heterogeneous Reasoning and Mediator System, 1994. http://www.cs.umd.edu/projects/hermes/overview/paper/index.html. 288

Global Semantic Serializability: An Approach to Increase Concurrency in Multidatabase Systems

Angelo Brayner[1] and Theo Härder[2]

[1] University of Fortaleza, UNIFOR, Dept. of Computer Science
60811-341 Fortaleza, Brazil
brayner@unifor.br
[2] University of Kaiserslautern, Dept. of Computer Science
D-67653 Kaiserslautern, Germany
haerder@informatik.uni-kl.de

Abstract. In this work, we present a new approach to control concurrency in multidatabase systems. The proposed approach is based on the use of semantic knowledge to relax the notion of absolute transaction atomicity. Supported by this new concept of atomicity, we propose a new correctness criterion, denoted global semantic serializability, for the execution of concurrent transactions, which provides *a high degree of inter-transaction parallelism*, *ensures consistency of the local databases* and *preserves autonomy of local databases*. Our proposal can also be used to increase concurrency in systems for integrating web data sources based on a mediator mechanism. Two concurrency control protocols we have developed are described.

1 Introduction

A multidatabase consists of a collection of autonomous databases, called local databases (LDBs). A key characteristic of local databases is that *they were created independently and in an uncoordinated way without considering the possibility that they might be somehow integrated in the future*. Systems used to manage multidatabases are called multidatabase systems (MDBSs). An MDBS should provide full database functionality and is built on top of multiple DBSs, called local DBSs (LDBSs), each of which operates autonomously. *Local autonomy* is a key feature of MDBS technology. The *multidatabase approach* provides inter-operability among multiple autonomous databases without attempting to integrate them by means of a single global schema [12].

A global application can access and update objects located in multiple databases by means of global transactions. In order to avoid inconsistencies, while allowing concurrent updates across multiple databases, MDBSs should provide a mechanism to control globally the concurrent access to local data.

Since the early seventies the concurrency control problem in a multiuser environment has been widely explored. In 1976, Eswaran *et al.* [8] proposed a model

© Springer-Verlag Berlin Heidelberg 2001

which introduces the concept of transaction. This model adopts a correctness criterion for the execution of concurrent transactions, called *serializability*, based on *transaction atomicity*. Serializability gives the illusion that the execution of a transaction is carried out in an isolated fashion without interference or interleaving from steps of other transactions. In other words, serializability gives the illusion that each transaction is executed as an atomic action. For that reason, we say that a transaction represents an atomic unit.

However, the nature and requirements of transaction processing in a multidatabase environment are quite different from those in conventional applications. Multidatabase transactions involve operations on multiple and autonomous local databases. Accordingly, they are relatively long-living. In addition, two different types of transactions may be executed in an MDBS context, global and local transactions. Although global and local transactions coexist, MDBSs do not have any information about the existence and execution order of local transactions due to local autonomy requirements. On the other hand, serializability requires knowledge of the execution order of all active transactions.

Therefore, the conventional concurrency control model is unsuited to MDBSs. To provide higher degree of inter-transaction parallelism in a multidatabase environment, new transaction models are needed, especially models that exploit multidatabase application semantics in order to relax serializability as a correctness criterion. Several researchers have started to extend serializability. However, the published solutions (as we will show in Section 3) either sacrifice local autonomy or support a low degree of parallelism.

The main motivation of this work is to provide an efficient solution to the concurrency control problem in multidatabase systems. We propose a new transaction model, denoted $\mathcal{GS}$-serializability, for synchronizing transactions in an MDBS environment. The proposed model *supports a high degree of parallelism among global transactions*, *ensures consistency of the local databases* and *preserves local autonomy of the local DBMSs*.

This work is structured as follows. Section 2 describes a model and reference architecture of MDBSs. Moreover, a running example which we use to illustrate definitions is presented. Some of the most important models for transaction processing in MDBSs are surveyed in Section 3. The new transaction model is presented in Section 4. Some realization aspects and the use of our approach in mediator based-systems are also discussed. In Section 5, two concurrency control protocols, each of which implementing a different approach to ensure $\mathcal{GS}$-serializability, are outlined. Section 6 concludes this paper.

2 The Multidatabase System Model

An MDBS integrates a set of pre-existing and autonomous local DBSs. In turn, each local DBS consists of a local DBMS and a database. Users interact with the local DBMS by means of transactions. Two classes of transactions are supported in a multidatabase environment:

- Local transactions which are transactions executed by a local DBMS outside the control of the MDBS and

- Global transactions which comprise transactions submitted by the MDBS to local DBMSs. A global transaction G_i consists of a set of subsequences $\{\text{SUB}_{i,1}, \text{SUB}_{i,2}, \text{SUB}_{i,3}, \ldots, \text{SUB}_{i,m}\}$ where each $\text{SUB}_{i,k}$ is executed at LDBS_k as an ordinary (local) transaction.

Observe that the notion of global transaction reflects the fact that subsequences are executed at different sites, which usually do not have direct communication.

Formally, An MDBS consists of:

1. a set $\mathcal{LD}=\{\text{LDBS}_1, \text{LDBS}_2, \ldots, \text{LDBS}_m\}$ of local database systems where $m > 1$;
2. a set $\mathcal{L}=\{\text{L}_1, \text{L}_2, \ldots, \text{L}_m\}$ of local transactions where each L_k represents the set of local transactions executed at the local system LDBS_k, with $0 < k \leq m$; and
3. a set $\mathcal{G}=\{\text{G}_1, \text{G}_2, \ldots, \text{G}_n\}$ of global transactions.

Operations belonging to global transactions are executed by local DBMSs. Local transactions result from the execution of local applications. Henceforth, a *global transaction* will be denoted by G and a *local transaction* by L.

A *local schedule* S_k models the execution of several interleaved operations belonging to local and global transactions performed at a particular local system LDBS_k. A *global schedule* S^G, on the other hand, models the execution of all operations executed by global and local transactions on the multidatabase.

The architecture of an MDBS basically consists of the Global Transaction Manager (GTM), a set of Interface Servers (servers, for short), and multiple local DBSs. To each local DBS, there is an associated server. A local DBS consists of a DBMS and at least one database. The GTM comprises three modules: Global Transaction Interface (GTI), Global Scheduler (GS), and Global Recovery Manager (GRM). An MDBS architecture is depicted in Figure 1.

3 Related Work

As already said, the conventional concurrency control model is unsuited to the MDBS technology. For that reason, several researchers started to extend the conventional transaction model or the concurrency control protocols based on serializability.

Du and Elmagarmid propose in [7] the quasi serializability model for the transaction processing in multidatabase environments. This model is based on the assumption that *update actions executed by global transactions on objects of a particular local database never depend in any way on the values of objects stored in other databases, which were previously read by the same transaction.*

A global schedule S^G is said to be *quasi serial* if *(i)* all local schedules are serializable *and (ii)* global transactions in S^G are executed serially such that for any two global transactions G_i and G_j the following is valid: if G_i precedes G_j in S^G, then all G_i's operations precede G_j's operations in all local schedules in which both transactions appear. In [7], it is shown that quasi serial schedules preserve multidatabase consistency.

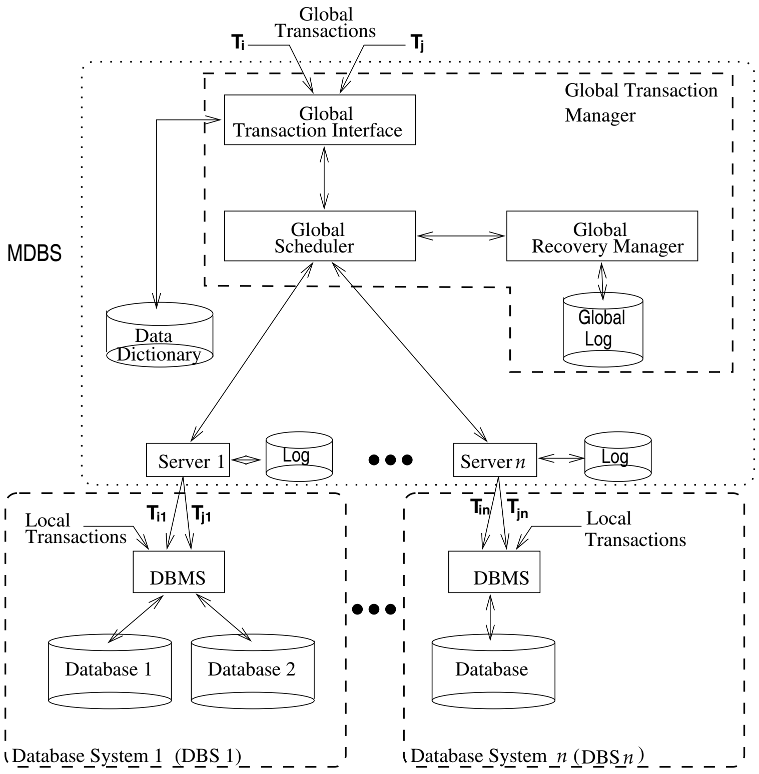


Fig. 1. A model for MDBS

The class of correct schedules is broadened by the notion of *quasi serializable* schedules. A global schedule is *quasi serializable* if it is conflict equivalent to a quasi serial schedule. In order to identify quasi serializable schedules, a graph-based method is proposed. The key idea of this method is to construct a directed graph, denoted *quasi serialization graph* (QSG), for a global schedule. In a QSG for a global schedule S^G ($QSG(S^G)$) the nodes represent the global transactions in S^G. The edges of a QSG reflect direct and indirect conflicts among global transactions. A global schedule S^G is quasi serializable if $QSG(S^G)$ is acyclic and all local schedules are conflict serializable.

This model relaxes global serializability. However, it still requires serializable execution of global transactions. Quasi serializability suffers additionally from the following problem. As seen, information about indirect conflicts among global transactions is needed in order to construct quasi serialization graphs. Indirect conflicts are provoked by the execution of local transactions. Only local systems have knowledge about the existence of local transactions. Consequently, information about indirect conflicts can only be provided by local systems. Hence, a GTM implementing quasi serializability presumes that local systems will provide information about local transactions. Clearly, *such information flow (local system to global system) violates local autonomy.* Recall that local autonomy is a key property in MDBS technology.

Mehrotra et al. [13] propose the *two level serializability* (2LSR) model which, according to the authors, relaxes global serializability. This model is based on the following assumptions:

- At each local database there are two types of stored data: *Local data* and *global data*.
- Local transactions may not modify global data. Hence, local transactions are restricted to execute write operations only on local data.

Considering the assumptions above, Mehrotra et al. define that a global schedule S^G is 2LSR if all local schedules are conflict serializable and the execution of global transactions in S^G is serializable.

In [2] we show that the 2LSR model represents, in fact, the application of the notion of predicatewise serializability [10,11] to multidatabase systems.

Since 2LSR is based on the notion of predicatewise serializability it inherits a serious shortcoming from the later model. 2LSR schedules may violate constraints. In [5] and [14] some examples are shown to illustrate this fact. Additionally, the 2LSR model presents the following two shortcomings. First, 2LSR assumes that objects in local databases are divided in local and global objects. Such an assumption represents a violation of local design autonomy since local database schemes should be modified in order to reflect the database division in local and global objects. Second, 2LSR requires that local transactions do not modify global objects. This strong restriction violates local execution autonomy.

The key problem for controlling concurrency in MDBS stems from the fact that global systems can not identify the serialization order of multidatabase transactions executed by local DBMSs. Georgakopoulos et al. [9] propose a strategy, denoted *ticket method*, to determine this order with the advantage that local systems do not need to give any information about the serialization order of transactions executing locally.

The basic idea of the ticket method is to force conflicts among multidatabase transactions. This is realized by the use of a special database object called *ticket*. Only one ticket is required per local system, and tickets may be accessed only by global transactions. Moreover, each subsequence of a global transaction executing at a local system must read the ticket value ($r(t)$), increment it ($t \leftarrow t + 1$), and update the new value into the local database ($w(t)$). The ticket method presumes that all local DBMSs ensure serializability and support *prepare-to-commit* operations.

Note that global transactions conflict when they try to access tickets. Such conflicts make it possible to determine the relative serialization order of subsequences of multidatabase transactions at each LDBS.

The ticket method requires that all global transactions access tickets. This may create a "hot spot" at the local database. Moreover, local database schemes should be altered in order to represent tickets. Some mechanism should be implemented at the local systems in order to ensure that only global transactions access tickets. Such requirements violate local autonomy.

4 The $\mathcal{GS}$-serializability Model

4.1 Basic Concepts

An MDBS integrates a collection of "pre-existing" local databases. Such local databases were created independently and in an uncoordinated way without considering that they will be integrated sometime in the future. For that reason, it is reasonable to see a multidatabase as a collection of disjoint sets of objects, each of which representing a single local database. We call those disjoint sets of objects *semantic units*. It is also reasonable to assume that the result of an update action executed by a global transaction on an object belonging to a particular semantic unit does not depend on the values of objects belonging to other semantic units which are previously read by the same transaction. Based on this semantic knowledge, we relax the notion of absolute transaction atomicity in order to provide a high degree of transaction concurrency in an MDBS environment. In our approach, a global transaction may consist of more than one atomic unit.

Before formalizing the notion of semantic units, we need to specify an additional concept denoted *depends-on*. This concept stems from the *dependence relation* between the (final) result of an updating operation on an object x and the value of another object y. We say that an object x depends-on an object y, if and only if the result of at least one update operation on x in any program (that accesses x and y) is a function of (i.e. is depending on) a value of y read in the same program. The set of all objects on which x *depends* is called depends-on-set(x).

Definition 1. *Let $\mathcal{DB}$ be a database. We say that SU_i, $0 < i \leq n$, are* semantic units *of $\mathcal{DB}$, iff*
(i) $\mathcal{DB} = \bigcup_{i=1}^{n} SU_i$,
(ii) $\forall 1 \leq i, j \leq n, i \neq j : SU_i \bigcap SU_j = \emptyset$, *and*
(iii) $(\forall x \in SU_i, y \in SU_j, i \neq j) \Rightarrow (x \notin \textit{depends-on-set}(y)) \wedge$
$(y \notin \textit{depends-on-set}(x))$ ◇

Intuitively, condition (iii) of Definition 1 reflects the idea that updates on objects of a semantic unit only depend on values of objects of the same semantic unit.

A *transaction* (global or local) is modeled as a finite sequence of *read* and *write* operations on database objects, where each object belongs to a particular semantic unit. We use $r_i(x)$ $(w_i(x))$ to represent a read (write) operation executed by a transaction T_i on a database object x. The set of operations executed by T_i is represented by $OP(T_i)$. In turn, $OP_{SU_a}(T_i)$ represents the set of operations belonging to T_i which are executed on objects of the semantic unit SU_a. Note that $OP_{SU_a}(T_i) \subseteq OP(T_i)$. It is assumed that *the execution of a transaction preserves database consistency if it runs isolated from other transactions.*

A transaction T is denoted module-structured if its operations are grouped into subsequences, called *modules*, such that each module represents an atomic unit of T. Intuitively, a *module-structured* transaction represents a sequence of modules where each module encompasses operations on objects of only one semantic unit. Further, the operations on objects of a semantic unit appear in only one module. For example, the following transaction may be characterized as being module-structured:

$$T_{Alice} = \underbrace{r_{Alice}(E)w_{Alice}(F)}_{module}\overbrace{r_{Alice}(P)w_{Alice}(U)w_{Alice}(V)}^{module}$$

Observe that the modules of a *module-structured* transaction are in fact atomic units. Here it is important to note that the notion of *module-structured* transactions reflects a transaction property. In other words, by means of this notion, we want to capture the fact that some transactions present a serial execution of atomic units without interleavings of operations belonging to different atomic units. It does not mean that our model requires that transactions should be partitioned into smaller pieces as proposed in [16].

Two schedules S_1 and S_2 over the set $\mathcal{T}= \{T_1, T_2, \cdots, T_n\}$ of transactions are said to be equivalent, denoted $S_1 \approx S_2$, if for any conflicting operations $p \in OP(T_i)$ and $q \in OP(T_j)$, the following condition holds: if $p <_{S_1} q$, then $p <_{S_2} q$. Observe that equivalent schedules produce the same effect on the database if they are executed on the same initial state.

Next, we define the concept of projection of a schedule on a set of transactions. Let S be a schedule over a set $\mathcal{G} \cup \mathcal{L}$ of transactions where $\mathcal{G}$ and $\mathcal{L}$ are disjoint sets of transactions. A projection $\mathcal{P}$ of S on the set $\mathcal{G}$ is a schedule for which the following conditions must hold:

(1) $\mathcal{P}$ only contains operations of transactions belonging to set $\mathcal{G}$
(2) $\forall p, q \in OP(\mathcal{P}) \Rightarrow p, q \in OP(S)$
(3) $\forall p, q \in OP(S'), p <_{\mathcal{P}} q \Leftrightarrow p <_S q$.

4.2 Correct Execution of Concurrent Transactions in MDBSs

In this section, we characterize correct schedules in our model. First, we define a standard for schedule correctness, denoted *global semantically serial schedules* ($\mathcal{GS}$-serial schedules, for short). Thereafter, we characterize schedules which produce the same effect on the database as a semantically serial one.

4.2.1 $\mathcal{GS}$-serial Schedules

Definition 2. *Let $S^G = \cup_{k=1}^{m} S_k$ be a global schedule over a set $\mathcal{T}= \mathcal{G} \cup \mathcal{L}$ of global and local transactions and $\mathcal{P}$ the projection of S^G on $\mathcal{G}$. The global schedule S^G is said to be $\mathcal{GS}$-serial if:*

(1) *each local schedule S_k is serializable and*
(2) *for each G_i in $\mathcal{P}$, G_i is module-structured and there is no interleaving within a module of G_i, for all modules in G_i (i.e., interleavings are only allowed between two modules of a transaction).*

We use $GS_eSerial$ to denote the class of all $\mathcal{GS}$-*serial* schedules over a given set of transactions. ◇

Intuitively, the latter condition of Definition 2 enforces that in a $\mathcal{GS}$-serial schedule the projection of S^G on $\mathcal{G}$ represents, in fact, a serial execution of modules belonging to multiple global transactions. This implies, the interleaving granularity for global transactions in a $\mathcal{GS}$-serial schedule is a module.

Theorem 1. *A $\mathcal{GS}$-serial schedule preserves multidatabase consistency.*
Proof. Let S^G be a $\mathcal{GS}$-serial schedule whose operations are performed on objects of a multidatabase $\mathcal{MDB}$.
<u>Case 1.</u> *Inconsistencies are caused by the execution of local schedules.*
Without loss of generality, suppose that inconsistencies are produced by the local schedule S_k at local database LDB$_k$. This is impossible, since, by Definition 2, each local schedule is serializable. Hence, the execution of every local schedule preserves database consistency, as was to be proved.
<u>Case 2.</u> *Inconsistencies are caused by the execution of $\mathcal{P}$.*
Without loss of generality, consider that the inconsistency results from operations executed on objects of semantic unit $SU_a \subseteq \mathcal{MDB}$. By assumption, the execution of $\mathcal{P}$ represents a serial execution of modules (second item of Definition 2). Hence, the execution of operations on objects of SU_a in $\mathcal{P}$ represents a serial execution of modules belonging to different transactions. Consequently, inconsistencies on objects of SU_a must have been produced by some module of a transaction in (recall that there is no interleaving within a module). Thus, the inconsistency must have been caused by the execution of some global transaction in $\mathcal{P}$. This is a contradiction because, by assumption, a transaction preserves database consistency. So, $\mathcal{P}$ cannot produce an inconsistent state. That is, $\mathcal{P}$ preserves database consistency, as was to be proved. ◇

4.2.2 $\mathcal{GS}$-serializable Schedules

So far, we have considered only $\mathcal{GS}$-serial schedules as being "safe". However, there are schedules which are not $\mathcal{GS}$-serial, but yield the same effect on the multidatabase as a $\mathcal{GS}$-serial one. That means, such schedules ensure multidatabase consistency and thereby may be considered as being "safe", too. Hence, we can broaden the class of safe schedules in our model and include these schedules.

Definition 3. *A global schedule $S^G = \cup_{k=1}^{m} S_k$ over a set $\mathcal{T} = \mathcal{G} \cup \mathcal{L}$ of global and local transactions is said to be $\mathcal{GS}$-serializable if and only if*
(1) *each local schedule S_k is serializable and*
(2) *the projection $\mathcal{P}$ of S^G on $\mathcal{G}$ is equivalent to the projection $\mathcal{P}_{SS}$ of a $\mathcal{GS}$-serial schedule S_{SS} over $\mathcal{T}$*

We denote the class of all $\mathcal{GS}$-serializable as GS_eSR. ◇

Intuitively, Definition 3 ensures that a global schedule S^G over a set $\mathcal{T}$ is $\mathcal{GS}$-serializable if and only if it is equivalent to a $\mathcal{GS}$-serial schedule over $\mathcal{T}$.

Another important benefit of $\mathcal{GS}$-serializability is that we can determine whether a global schedule is $\mathcal{GS}$-serializable by verifying the acyclicity of a directed graph, called *semantic serialization graph* (S_eSG).

Definition 4. *Let $S^G = \cup_{k=1}^m S_k$ be a global schedule over a set $\mathcal{T} = \mathcal{G} \cup \mathcal{L}$ of global and local transactions and $\mathcal{P}$ the projection of S^G on $\mathcal{G}$. The* Semantic Serialization Graph *for $\mathcal{P}$ is a directed graph $S_eSG(\mathcal{P}) = (N, E)$. The set N of nodes represents the transactions in $\mathcal{G}$, i.e., $N = \mathcal{G}$. The set E represents labeled edges of the form $G_i \xrightarrow{SU_k} G_j$, where:*

- *$G_i, G_j \in N$ and*
- *there are two operations $p \in OP(G_i), q \in OP(G_j)$, $p <_P q$, on an object of the semantic unit SU_k, which are in conflict.* $\diamond$

Lemma 1. *If a schedule $S^G = \cup_{k=1}^m S_k$ is $\mathcal{GS}$-serial, then the* Semantic Serialization Graph *for the projection $\mathcal{P}$ of S^G on $\mathcal{G}$ is acyclic.*

Proof. Let $S^G \in \mathrm{GS}_e\mathrm{Serial}$ be a schedule over the set $\mathcal{T} = \mathcal{G} \cup \mathcal{L}$ of global and local transactions, where $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$. By condition 2 of Definition 2, the projection $\mathcal{P}$ of S^G over $\mathcal{G}$ represents a serial execution of modules of each transaction $G_i \in \mathcal{G}$, $0 < i \leq n$. Suppose, by way of contradiction, that $S_eSG(\mathcal{P})$ is cyclic and, without loss of generality, the cycle has the following form: $G_i \xrightarrow{SU_n} G_j \xrightarrow{SU_n} \cdots \xrightarrow{SU_n} G_i$. It follows from this that the *module* of G_i which represents the operations of G_i on objects of the semantic unit SU_n is interleaved by some operations of G_j, a contradiction, by assumption, S satisfies condition 2 of Definition 2. Therefore, $S_eSG(\mathcal{P})$ is acyclic, as was to be proved. $\diamond$

Theorem 2. *A global schedule $S^G = \cup_{k=1}^m S_k$ over a set $\mathcal{T} = \mathcal{G} \cup \mathcal{L}$ of global and local transactions is gs-serializable if and only if:*

1. *the serialization graph (see [1]) for each local schedule S_k, $0 < k < m$, is acyclic, and*
2. *the semantic serialization graph for the projection of S^G on set $\mathcal{G}$ of global transactions is acyclic.*

Sketch of Proof.

Condition 1. By Definition 3, a schedule is gs-serializable if each local schedule S_k is serializable. In [1], it is shown that a schedule is serializable *if and only if* its serialization graph is acyclic. Hence, the *serialization graph* for each local schedule S_k is acyclic.

Condition 2. (“$\longrightarrow$”) In a $\mathcal{GS}$-serializable schedule, the projection of S^G on $\mathcal{G}$ is equivalent to a $\mathcal{GS}$-serial schedule. By Lemma 1, the semantic serialization graph for the projection of a $\mathcal{GS}$-serial schedule Se^G on set $\mathcal{G}$ is acyclic. Since S^G is equivalent to Se^G (a $\mathcal{GS}$-serial schedule), the semantic serialization graph for the projection of S^G on set $\mathcal{G}$ is acyclic, too.

(“$\longleftarrow$”) Consider that the semantic serialization graph $S_eSG(\mathcal{P})$ for the projection of S^G on set $\mathcal{G}$ contains edges with label SU_n and it is acyclic. Thus, we may topologically sort it by edges with label SU_n. However, for this topological sort, instead of considering a transaction G_i as a node, we consider only the module of G_i, which contains the operations of G_i over objects of the semantic unit SU_n. As a result of this “modified” topological sort, we obtain a serial execution of modules of transactions in $\mathcal{G}$, where this serial execution contains operations on objects of the semantic unit SU_n. If we repeat this process “recursively” for each label in $S_eSG(\mathcal{P})$, we obtain a serial execution of modules of all global transactions in $\mathcal{G}$. Moreover, there is no interleaving within each module. We have, thus, a $\mathcal{GS}$-serial schedule (by Def. 2) which we call S_{SS} over the set $\mathcal{G}$ of global transactions. Since the projections $\mathcal{P}$ of S^G on set $\mathcal{G}$ and $\mathcal{P}_{SS}$ of S_{SS} on $\mathcal{G}$ have the same set of operations and order the conflict operations in the same way, they are equivalent. Therefore, by Definition 3, S^G is $\mathcal{GS}$-serializable, as was to be proved. $\diamond$

Theorem 2 has a very important practical impact. If we assume that all participating local DBMSs enforce *syntactic* serializability, we only need to verify the acyclicity of the semantic serialization graph for the execution of global transactions. This is a quite reasonable assumption, since all existing database systems implement serializability. Hence, we can apply this strategy to control concurrency in a multidatabase system.

Remark 1. Let $\mathcal{M}$ be a multidatabase system, where:

(i) all participating local DBMSs enforce serializability as local correctness criterion for schedules and

(ii) the GTM of $\mathcal{M}$ implements semantic serializability to synchronize global transactions.

Consider a schedule S^G over a set $\mathcal{G} \cup \mathcal{L}$ of global and local transactions. Furthermore, the operations of S^G are executed by $\mathcal{M}$. The schedule S_{GTM} represents the projection of S^G on set $\mathcal{G}$. *The acyclicity of the graph $S_eSG(S_{GTM})$ is the necessary and sufficient condition to determine whether or not S^G is gs-serializable (correct).* $\diamond$

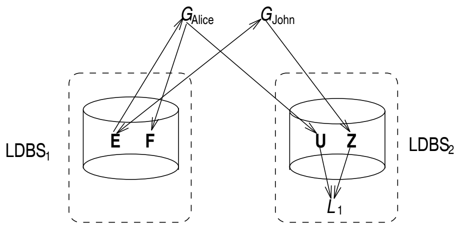
Remark 1 ensures that the transaction manager component of an MDBS can determine the correctness of global schedules without receiving any kind of information from the local systems. That means, a GTM implementing gs-serializability does preserve local autonomy. Recall that some of the proposals examined in Section 3 violate local autonomy.

Example 1. Consider a global schedule S^G which is executed in a multidatabase application. The execution scenario for S^G is depicted in Figure 2.

By Definition, $S^G = S_{LDBS_1} \cup S_{LDBS_2}$. The projection of S^G on the set of global transactions is represented in Figure 2 by the schedule S_{GTM}. By Theorem 2, the global scheduler S^G is correct, since the semantic serialization graph for S_{GTM} is acyclic (Figure 3) and the serialization graphs for the local schedules S_{LDBS_1} and S_{LDBS_2} contain no cycles.

However, if we had assumed that the two local systems of the multidatabase application of Figure 2 enforce serializability, we could have considered Remark 1. In this case, only the acyclicity of $S_eSG(S_{GTM})$ had to be verified. This procedure had already indicated that the global schedule S^G is correct, without violating local autonomy. Observe that S^G is not quasi serializable.

Schedule S^G could also not be produced by concurrency control mechanisms implementing the *ticket method* (TM) or *altruistic locking* (AL) [15] protocols. That is because S^G is not executed in a serializable fashion. $\diamond$



$G_{Alice} = r_{G_{Alice}}(E)w_{G_{Alice}}(U)w_{G_{Alice}}(F)$
$G_{John} = r_{G_{John}}(E)w_{G_{John}}(E)w_{G_{John}}(Z)$
$L_1 = r_{L_1}(U)r_{L_1}(Z)$
$S_{LDBS_1} = r_{G_{John}}(E)w_{G_{John}}(E)r_{G_{Alice}}(E)w_{G_{Alice}}(F)$
$S_{LDBS_2} = w_{G_{Alice}}(U)r_{L_1}(U)r_{L_1}(Z)w_{G_{John}}(Z)$
$S^G = r_{G_{John}}(E)w_{G_{John}}(E)r_{G_{Alice}}(E)w_{G_{Alice}}(U)r_{L_1}(U)w_{G_{Alice}}(F)r_{L_1}(Z)w_{G_{John}}(Z)$
$S_{GTM} = r_{G_{John}}(E)w_{G_{John}}(E)r_{G_{Alice}}(E)w_{G_{Alice}}(U)w_{G_{Alice}}(F)w_{G_{John}}(Z)$

Fig. 2. Execution scenario of Example 1

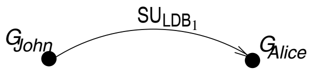


Fig. 3. Semantic serialization graph for the schedule S_{GTM}

Figure 4 depicts the relationship between the class of $\mathcal{GS}$-serializable schedules and some classes of schedules.

In order to show that the class of QSR schedules is a subset of GS_eSR, consider the schedule S^G depicted in Figure 2. As seen in Example 1, S^G does not belong to the class of QSR schedules. However, S^G is an element of GS_eSR. This implies, $GS_eSR \setminus QSR \neq \emptyset$. Recall that a schedule S belongs to the class QSR if the global transactions in S are executed in a serializable fashion. Schedules belonging to GS_eSR do not necessarily present a serializable execution of their global transactions, since gs-serializability relaxes classical serializability. Therefore, $QSR \subset GS_eSR$.

4.3 Realization Aspects

In order to implement $\mathcal{GS}$-serializability for controlling concurrency in MDBSs, each participating local database should be defined as a semantic unit. Hence, the acquisition of information about the precise locality of database objects is a key question for using $\mathcal{GS}$-serializability in the multidatabase technology. However, such a problem is already addressed by MDBSs, since global systems must identify where global operations are to be performed.

Alternatively, we propose a mechanism which enables the GTM to automatically identify the location of database objects and thereby to identify the correct

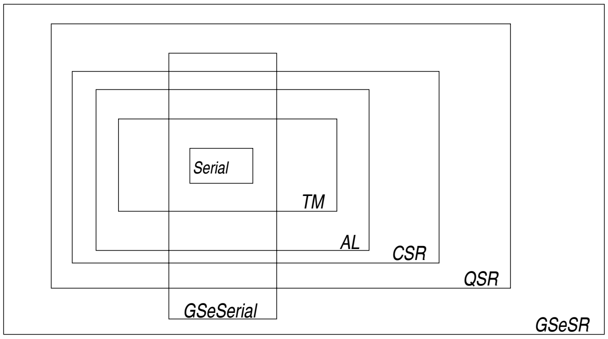


Fig. 4. Relationship among different classes of schedules

specification of semantic units. The basic principle of this mechanism is to use the component *Data Dictionary* of the MDBS architecture depicted in Figure 1. By doing this, sufficient information about local databases can be stored in the data dictionary. This information may be used by the GTM, more specifically by the GTI, in order to determine where each global operation should be executed. The GTI may forward this information to the GS. Such an information flow will give the GS the sufficient and necessary support to correctly determine the semantic units and their corresponding database objects.

Note that the process for identifying semantic units is realized without intervention of users, all is performed automatically.

4.4 Extending the Notion of Semantic Units in MDBSs

The notion of semantic unit is flexible enough to allow that two or more local databases can be logically grouped in order to represent a single semantic unit, without violating local autonomy of each local system. By "logically", we mean that only the GTM should be aware of such a representation. It is not necessary to physically join neither the databases nor the database systems.

Specifying more than one local database as a single semantic unit enables our transaction-processing model to synchronize transactions in MDBSs which have the following characteristics:

(i) Data replication. Some MDBSs may contain objects replicated in more than one local database. In this case, the local databases containing replicated data should be grouped in a semantic unit;

(ii) Global constraints. Constraints which span more than one local database are called global constraints. In this case, local databases containing objects referred in a global constraint should determine a semantic unit.

4.5 Increasing Concurrency in Mediator-Based Systems

Mediator-based systems have been widely used to integrate heterogeneous web data sources. In such systems, wrappers are responsible for converting local data into a common model. In turn, a mediator provides an integrated view over the data exported by wrappers.

The integrated view can be either virtual or materialized. In the virtual approach, queries submitted to the mediator are decomposed into sub-queries, which are executed on the local web data sources. In other words, queries submitted to the mediator represent global transactions. A global transaction consist of a set of subsequences, where each subsequence corresponds to a subquery executed at a local web source as an ordinary (local) transaction.

In order to implement $\mathcal{GS}$-serializability to control concurrency in mediator-based systems (for integrating web data sources), each web source can be defined as a semantic unit. Since the mediator provides an integrated view of web data, the mediator can automatically identify the location of database objects and thereby identify the correct specification of semantic units.

5 Concurrency Control Protocols

5.1 Semantic Locking (S_eL)

The S_eL protocol associates a lock to each database object. Three types of locks are supported: read-only, write and update locks. A transaction accesses an object if and only if a lock can be associated to the object on behalf of the transaction. Another key characteristic presented by the S_eL protocol is to implement the two-phase property of the conventional 2PL protocol. However, it uses another granularity for realizing the two-phase property. In the 2PL protocol, this granularity is a transaction, since once a transaction has released a lock it may not obtain another lock. In the S_eL protocol, the granularity is represented by the subsequence of operations on objects of one local database. This implies, locks held by a global transaction G on objects of local database LDB_k may be released after the last operation of G on objects of LDB_k.

Therefore, the S_eL protocol guarantees that locks may be released by a global transaction before they complete their executions. This property increases the concurrency among global transactions. Moreover, local DBMSs do not need to hold locks on local resources on behalf of global (and remote) transactions for a long period of time. Additionally, the S_eL protocol presents the following benefits. First, it reduces the frequency of deadlocks caused by lock conversions. Second, it implements a variable granularity locking strategy. Multiple lockable units support that concurrency may be enhanced by fine granularity, or locking overhead may be reduced by coarse granularity.

5.2 The S_eSG Checking Protocol

The protocol, denoted S_eSGC, is based on a similar strategy which is used by the conventional serialization graph testing protocol [6]: *the dynamic monitoring and* *management of an always acyclic conflict graph*. In contrast to the classical serialization graph testing, an S_eSGC protocol exploits semantic knowledge provided by the notion of semantic units.

The graph maintained by the S_eSGC protocol is called semantic conflict graph ($\mathcal{SC}$-graph). It is constructed according to the same rules used to construct a semantic serialization graph (Definition 4). Hence, nodes of the $\mathcal{SC}$-graph represent transactions and edges reflect conflicts between transactions. Notwithstanding, a $\mathcal{SC}$-graph differs from semantic serialization graphs in two aspects. First, not all committed transactions must be represented. Second, not all conflicts must be represented as an edge of the $\mathcal{SC}$-graph. That is because, in some cases, nodes and edges may be "safely" removed from the $\mathcal{SC}$-graph. Later we will show how this can be done.

The protocol works as follows. When a global scheduler (GS) using the S_eSGC protocol starts running, the $\mathcal{SC}$-graph is created as an empty graph. As soon as the scheduler receives the first operation of a new transaction (*begin-transaction*) G_i, a node representing this transaction is inserted in $\mathcal{SC}$-graph. For each operation $p_i(x) \in OP(G_i)$ which the GS receives, it checks if there is a conflicting operation $q_j(x) \in OP(G_j)$ which has already been scheduled. If an operation $q_j(x)$ has already been scheduled, the scheduler inserts an edge of the form $G_j \overset{SU_{LDB_k}}{\longrightarrow} G_i$, where x is an object belonging to the semantic unit SU_{LDB_k}. In fact, x is an object of the local database LDB_k.

Thereafter, the GS verifies if the new edge introduces a cycle in the $\mathcal{SC}$-graph. In the affirmative case, the GS rejects the operation $p_i(x)$, undoes the effect of operations of the subsequence $SUB_{i,k}$ and removes the edge $G_j \overset{SU_{LDB_k}}{\longrightarrow} G_i$ from $\mathcal{SC}$-graph. Otherwise, $p_i(x)$ is accepted and submitted to the corresponding server.

If the global scheduler identifies a cycle in the $\mathcal{SC}$-graph, only operations belonging to the atomic unit whose operation provokes the cycle are to be rolled back. It is not necessary to abort the entire global transaction.

6 Conclusions

In order to fulfil the requirements of transaction processing in MDBSs we have introduced a new transaction processing model. The key principle behind the proposed model is the use of semantic knowledge which is captured by means of the notion of *semantic units*. By means of the concept of semantic units, absolute transaction atomicity can be relaxed. Supported by this new notion of atomicity we have proposed a new correctness criterion, denoted $\mathcal{GS}$-serializability, for the execution of concurrent transactions in MDBSs. We have shown that $\mathcal{GS}$-serializability enforces multidatabase consistency, provides a high degree of inter-transaction parallelism, while preserving local autonomy (since it does not require any information about the execution of global transactions at the local systems). The notion of semantic units can be extended to allow two or more local databases to determine a single semantic unit.

Finally, two concurrency control protocols based on $\mathcal{GS}$-serializability were described. Although we have already developed a recovery mechanism for MDBSs using $\mathcal{GS}$-serializability (we refer the reader to [3]), we are aware that we have to investigate further on this direction. We are now working on the problem of global deadlock detection and resolution.

References

1. Bernstein, P. A., Hadzilacos, V. and Goodman, N. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987. 309
2. Brayner, A. *Transaction Management in Multidatabase Systems.* Shaker-Verlag, 1999. 305
3. Brayner, A. and Härder, T. Recovery in multidatabase systems. In *Procedings of XIV Brazilian Symposium on Databases (SBBD 99)*, 1999. 315
4. Brayner, A., Härder, T. and Ritter, N. Semantic Serializability: A Correctness Criterion for Processing Transactions in Advanced Database Applications. *Data & Knowledge Engineering*, 31(1):1–24, 1999.
5. Breitbart, Y., Garcia-Molina, H., Silberschatz, A. Overview of multidatabase transaction management. *The VLDB Journal*, (2):181–239, 1992. 305
6. Casanova, M. A. The Concurrency Problem of Database Systems. In *Lectures Notes in Computer Science*, number 116. Springer-Verlag, 1981. 313
7. Du, W. and Elmagarmid, A. K. Quasi Serializability: a Correctness Criterion for Global Concurrency Control in InterBase. In *Proceedings of the 15th International Conference on VLDB*, pages 347–355, Amsterdam, 1989. 303
8. Eswaran, K. P., Gray, J. N., Lorie, R. A. and Traiger, I. L. The Notions of Consistency and Predicate Locks in a Database System. *Communications of the ACM*, 19(11):624–633, November 1976. 301
9. Georgakopoulos, D., Rusinkiewicz, M. and Sheth, A. Using Tickets to Enforce the Serializability of Multidatabase Transactions. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):1–15, February 1993. 305
10. Korth, H. F. and Speegle, G. D. Formal Model of Correctness Without Serializability. In *Proceedings of ACM SIGMOD Conference*, pages 379–386, 1988. 305
11. Korth, H. F. and Speegle, G. D. Formal Aspects of Concurrency Control in Long-Duration Transaction Systems Using The NT/PV Model. *ACM Transactions on Database Systems*, 19(3):492–535, September 1994. 305
12. Litwin, W., Mark, L. and Roussopoulos, N. Interoperability of Multiple Autonomous Databases. *Computing Surveys*, 22(3):267–293, 1990. 301
13. Mehrotra, S., Rastogi, R., S., Korth, H. and Silberschatz, A. Non-serializable Executions in heterogeneous distributed database systems. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, 1991. 305
14. Rastogi, R., Mehrotra, S., Breitbart, Y., Korth, H. and Silberschatz, A. On Correctness of Non-serializable Executions. In *Proceedings of the SIGMOD PODS*, pages 97–108, 1993. 305
15. Salem, K., Garcia-Molina, H. and Shands, J. Altruistic Locking. *ACM Transactions on Database Systems*, 19(1):117–165, March 1994. 310
16. Shasha, D., Simon, E. and Valduirez, P. Simple Rational Guidance for Chopping Up Transactions. In *Proceedings of 1992 ACM SIGMOD Conference*, pages 298–307, 1992. 307

Checking Integrity Constraints in Multidatabase Systems with Nested Transactions*

Anne Doucet[1], Stephane Gançarski[1], Claudia León[1,2], and Marta Rukoz[2]

[1] LIP6, Case 169, Université P&M Curie
4, place Jussieu, 75252 Paris. France.
{Anne.Doucet,Stephane.Gancarski,Claudia.Leon}@lip6.fr
[2] CCPD, U.C.V. Apdo.
47002, Los Chaguaramos, 1041A. Caracas. Venezuela.
mrukoz@kuaimare.ciens.ucv.ve

Abstract. This paper proposes various strategies for the checking of integrity constraints in multidatabase systems which support nested transactions. The solution presented in [8] for centralized environments is extended. The principle of this solution consists of designating a sub-transaction which controls the checking of each integrity constraint. This sub-transaction is the smallest common ancestor (within the nested transaction tree) of all the sub-transactions which might violate the constraint. In the case of a multidatabase, it is necessary to take into account the constraint structure and the localization of the sub-transactions, to choose the site where the checking should be performed in order to minimize data transfers through the network. For this purpose, different checking strategies are presented depending on the type of the constraint to be checked.

Keywords: Integrity constraints. Nested transactions. Multidatabase systems. Consistency checking. Distributed transactions.

1 Introduction

An important functionality in databases (DB) is to maintain consistency. This functionality is generally ensured by the definition of integrity constraints (IC) which are logical assertions which must be satisfied at the end of each transaction. Much work has been devoted to this problem (see [9] for a good survey) and most commercial DBMS provide efficient tools to maintain consistency. However, few studies have been carried out for the handling of integrity constraints in multidatabase systems. In such systems, it is necessary to minimize the transfer of data among the involved sites, and thus to determine on which site(s) the constraints should be checked. Some works have studied the problem of constraint checking in federated databases [6,12], distributed databases [12], and multidatabase systems [11] but none of them considers the nested transactions model

* This work was supported by CNRS in France and by CONICIT and CDCH-UCV in Venezuela

© Springer-Verlag Berlin Heidelberg 2001

as a base for the users applications, although this model is well adapted to these systems. [16] presents a framework to manage consistency among interdependent data in a multidatabase environment. In this approach, when a transaction is sent to a local database, a set of related transactions should be scheduled by the system in order to preserve consistency.
Since they were proposed by Moss [13], nested transactions (NT) have been the subject of many studies. Research prototypes [5,14] and products such as Versant [3] and Encina [1] supporting nested transactions have been developed. However, few works address the issue of integrity constraint checking in the context of nested transactions. In [7], the checking of ICs in centralized systems with NT is studied. ICs are defined at the method level. In this approach, the programmer must describe the actions to perform when a constraint is violated and at which level of a NT those actions must be executed. In [8], a mechanism for handling ICs in database systems that support NT is described. ICs are defined in a global way at the DB schema level. A syntactic analysis allows to detect the objects manipulated by the constraints and touched by the transaction. The code for the constraint checking is automatically generated and inserted.
Maintaining consistency in multidatabase systems raises new problems. As data is stored on several sites, it is essential to determine where each constraint must be defined, stored and checked. The choice of an appropriate site must take into account the constraint type, the kind of update, and the sites where the updates are performed. In all cases, data transfers should be minimized. In this paper, we present a solution to check global ICs (which involve data on several sites) for nested transactions in multidatabase systems. We extend the approach described in [8] by considering data distribution.
This paper is organized as follows. Section 2 shortly describes different approaches for checking ICs and introduces the principle used in this work. Section 3 briefly presents the mechanism proposed in [8] for the checking of ICs in the context of NTs. In Section 4, after describing the architecture of the system that we consider, we present the main problems due to data distribution. In Section 5 we present different strategies to carry out the checking of ICs, according to the constraint type and the localization of the sub-transactions which touch it. Finally, in Section 6 we conclude and present some perspectives.

2 Integrity Constraints Checking

A database is consistent if and only if all integrity constraints are satisfied. However, the checking of all constraints after each transaction has a prohibitive cost. Numerous works have studied this problem [9]. A popular approach consists of checking constraints at the end of the transaction, using compilation techniques to reduce the checking time at execution [4,2]. Our work is based on this approach and therefore can be adapted to all constraint handling mechanisms provided by a DBMS using the following principles:

- ICs are declared in a global and declarative way.
- A syntactic analysis of both constraints and transactions reduces the set of constraints to check, since it allows to determine the set of constraints that might be violated by a transaction. In this paper, we consider that the syntactic analysis determines the objects *involved* in a constraint and the objects *touched* by a transaction. A transaction might violate a constraint if it touches objects involved in the constraint.
- Constraint checking is automatically performed. To optimize the cost of the checking process, DBMSs generally provide different constraint checking algorithms, adapted and optimized according to the different types of constraints. An other classic optimization consists of reducing the set of objects for which a constraint must be checked. A constraint is checked only with respect to the objects involved in the constraint and modified by the transaction. These objects are determined and collected at execution time. Clearly, this optimization, which limits the checking to a reduced set of objects, only concerns universally quantified constraints. Our approach integrates these two standard optimizations.

3 Nested Transactions and Integrity Constraints Checking

Nested transactions are appropriated to multidatabase systems where long transactions can be broken down into several sub-tasks [1,5,14]. A nested transaction (NT) is a tree of transactions of an arbitrary depth where the components are sub-transactions. The transaction on top of the tree is the *transaction root*. Transactions having sub-transactions are called *parents*, and their sub-transactions are their *children*. Transactions that do not have sub-transactions are called *leaves*. Each sub-transaction controls the execution of its child sub-transactions. It begins before its children and finishes after them. Each sub-transaction is executed independently and possibly in parallel with other sub-transactions. This means that it can decide to commit or to abort independently. This decision may depend on the result of the execution of its sub-transactions. If a sub-transaction aborts, its sub-transactions must abort, but this does not necessarily require its parent to abort. If a sub-transaction decides to commit, the validation is not definitive, since its updates will only be performed if all its ancestor transactions decide to commit (i.e. only if the root transaction commits). When the root transaction commits, only updates of all sub-transactions which decided to commit and which have no aborted ancestor, will be taken into account. An NT may thus complete, while maintaining consistency, even if some of its sub-transactions have aborted.
The solution presented in [8] guarantees that if the root transaction commits, the database will be in a consistent state. We summarize below the main characteristics of this solution, for more details the reader can refer to [8]. Let T be a nested transaction executed on a database associated to a set of constraints. The main idea of this approach is the selection, for each constraint, of a sub-

transaction responsible for its checking : the smallest common ancestor of the sub-transactions that touch the constraint. To this aim, the analysis techniques described in the previous section are used. It is thus possible to determine at compile time:

– For each leaf sub-transaction Tl_j of a nested transaction T, the set C_j of all the constraints touched by Tl_j.
– For each constraint C_i, the set $T(C_i)$ of leaf sub-transactions Tl_j touching C_i. We will denote $SCA(C_i)$ the smallest common ancestor of all the sub-transactions in $T(C_i)$.

$SCA(C_i)$ is responsible for the maintenance of constraint C_i. As this node controls the execution of its sub-tree, it is possible to ensure, in case of violation of C_i, that not only the sub-transactions which violated C_i will abort, but also all the sub-transactions that have used its results. As opposed to conventional integrity mechanism systems for flat transactions, the sub-transactions that do not touch C_i can continue their execution, allowing thus to limit the number of aborts. The direct communication between the leaf sub-transactions that touch a constraint and the sub-transaction responsible for its checking, allows to initiate the checking process ($check(C_i)$), as soon as all the leaf transactions that touch the constraint C_i have finished their execution. In case of violation, the propagation of an abort message downwards the corresponding sub-tree, allows to abort the sub-transactions which must be undone to restore the satisfaction of the constraint.

4 Global Constraints in a Multidatabase Environment

A multidatabase system supports operations on several databases, each one handled by a local database management system (DBMS). Those systems may have different architecture and integration levels corresponding to different levels of global services [15]. Our objective is to maintain global integrity constraints on a multidatabase system. These constraints touch objects located on different sites. In this paper, we consider a multidatabase system with the following characteristics:

1. All transactions are global and managed by the global transaction manager (GTM) by using a global scheme that describes each object and its localization in the system. This manager guarantees the concurrency control among transactions and among sub-transactions of a transaction (see [17]).
2. We use a nested transactions model where only leaf transactions can perform updates. Each leaf is totally executed on a single site and only accesses objects on that site. This allows to simplify the constraint checking process and does not decrease the expressiviness of the model (see [13]).

The assignment of the sub-transactions of a NT to the different sites is induced by the localization of the objects in the network. When a global transaction is initiated on a site, its sub-transactions are recursively initiated on the same site, until a sub-tree containing leaves that are all executed on a same site is found. This subtree is then sent to the corresponding site and is completely executed on this site. Figure 1 illustrates this operation where the global transaction T is initiated on S_i. Sub-transactions T_1 and T_2, which have leaves of their sub-trees on different sites, will be executed on the initial site S_i. Sub-transaction T_{11}, child of T_1, has all the leaves of its sub-tree on site S_1. T_{11} is, therefore, sent to site S_1 where its whole sub-tree will be executed. Note that a sub-tree assigned to a site can be a single leaf sub-transaction, as it is the case for Tl_{12} and Tl_{21}.

The distribution of objects and sub-transactions on different sites implies to

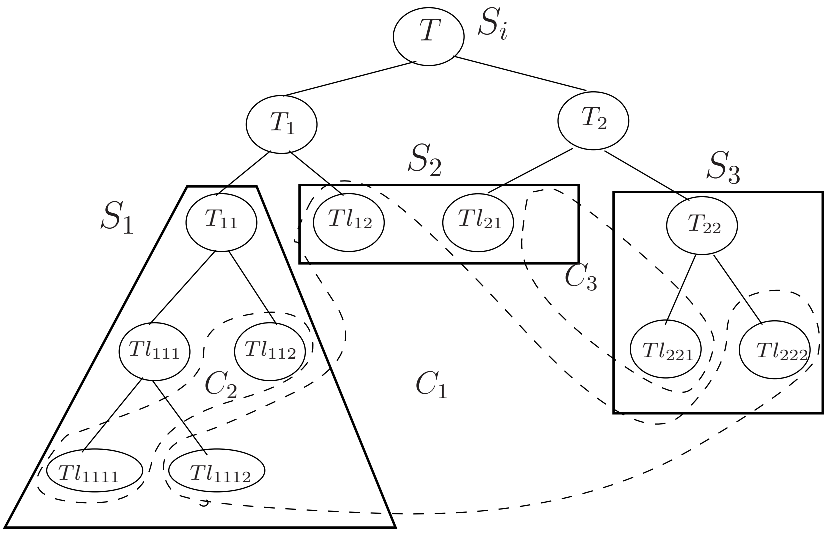


Fig. 1. Global constraint touched by a distributed nested transaction

extend the solution presented in Section 3 for integrity constraint checking. The main principle remains the same: the smallest common ancestor $SCA(C_i)$ of the leaf sub-transactions touching constraint C_i *controls* the constraint checking. Note that $SCA(C_i)$ is located either on the site where all leaf sub-transactions touching C_i are performed, or on the site where the global transaction is initiated. Nevertheless, this doesn't mean that the checking of constraint C_i, which is performed by the process $Check(C_i)$, is executed on the site where $SCA(C_i)$ is located. $Check(C_i)$ is in fact a sub-transaction of $SCA(C_i)$ which has a child on each site where C_i (or a part of it) has to be checked. Indeed, for sake of efficiency, $Check(C_i)$ must take into account the communication costs among the sites to minimize the checking time and is performed on the most appropriated site(s). Two cases can be distinguished:

1. Constraint C_i is local : it involves objects stored on the same site S_j. In this case, the most efficient solution is obviously to perform $Check(C_i)$ on site S_j.
2. Constraint C_i is global : it involves data located on different sites. In this case, it is necessary to choose the site(s) where to execute the checking

process, trying to minimize the number of messages to send on the network, as well as the size of these messages.

To efficiently check constraints on a multidatabase system supporting nested transactions, it is necessary to consider both the type of the constraints and the structure of the nested transaction, in particular the location of the leaf sub-transactions. This problem is illustrated on Figure 1. The objects involved in each constraint C_1, C_2 or C_3 are in the domain delimited by dashed lines (the name of the constraint being inside the domain). For example, C_1 involves data on sites S_1, S_2 and S_3 and it is touched by Tl_{1112} on S_1, by Tl_{12} on S_2 and by Tl_{222} on S_3. C_2 is a local constraint on S_1 and it is touched by Tl_{112} and Tl_{1111} on that site. This example shows that the definition a checking strategy may be complex:

- For constraint C_3, although the only leaf sub-transaction that touches the constraint is executed on site S_3, the checking process should be executed mainly on site S_2 if C_3 is a universal constraint. Indeed, in this case, the evaluation of C_3 requires to compare the objects modified by Tl_{221} on S_3 with the objects associated to the constraint, located on S_2. To evaluate the constraint on S_2, it is only necessary to transfer to S_2 the objects modified on S_3. The constraint evaluation on S_3 would require to transfer all the objects of S_2 involved by the constraint, including those that are not directly or indirectly affected by the transaction.
- For constraint C_1, the situation is even more complex. The constraint evaluation can be performed on S_1, S_2, S_3 or on any combination of those sites, according to the nature of the constraint and to the actions performed by Tl_{1112}, Tl_{12} and Tl_{222}.

5 Checking Strategies of Global Constraints

Determining the best site(s) where to perform the checking of a global constraint depends on the type of the constraint. In this section we give a classification of global constraints and for each type, we propose an appropriate checking strategy.

5.1 Global Constraints Locally Checkable

Global constraints that can be expressed by a conjunction or a disjunction of local constraints can be checked using independent local processes.

• **Conjunction of local constraints :** Global constraints that can be expressed by a conjunction of local constraints are of the following form: $C_1 \wedge C_2 \ldots \wedge C_n$ such that, for all i, C_i is a local constraint of the database of site S_i. If all C_i are completely independent (i.e. if they touch different classes of objects and are not related), global constraint C can be re-written, during the constraint design phase, as the conjunction of the C_i. For example, if C is $\forall o_1 \in Cl_1, \forall o_2 \in$

$Cl_2, F(o_1, o_2)$, with class Cl_1 (resp. Cl_2) on site S_1 (resp. S_2) and if $F(o_1, o_2)$ can be re-written as $F_1(o_1) \wedge F_2(o_2)$, then two independent constraints can be declared, $C_1 \;:\; (\forall o_1 \in Cl_1, F_1(o_1))$ on site S_1 and $C_2 \;:\; (\forall o_2 \in Cl_2, F_2(o_2))$ on site S_2. In some cases, the decomposition of C must take into account data fragmentation. For example, if constraint C is : $\forall o \in Cl, F(o)$ and if class Cl is fragmented on $S_1, \ldots, S_n$, then constraint C becomes $(\forall o_1 \in Cl_{S_1}, F(o_1)) \wedge (\forall o_2 \in Cl_{S_2}, F(o_2)) \ldots \wedge (\forall o_n \in Cl_{S_n}, F(o_n))$, where Cl_{S_i} is the fragment of Cl located on S_i.

Checking strategy: The checking strategy for this type of constraints is simple. During compilation, C is re-written as $C_1, \ldots, C_n$. C is satisfied if and only if the set of C_i are satisfied. Maintaining C is reduced to maintain each C_i, which can be performed by the DBMS of the corresponding site.

• **Disjunction of local constraints :** Global constraints that can be expressed as a disjunction of local constraints are of the form : $C_1 \vee C_2 \ldots \vee C_n$ such that C_i is a local constraint on site S_i. Such constraints are, for example, existential constraints on a fragmented class. Thus, a constraint of the form $\exists o \in Cl, F(o)$ with Cl fragmented in $Cl_1, Cl_2, \ldots Cl_n$ on sites $S_1, S_2, \ldots S_n$ respectively becomes $(\exists o_1 \in Cl_1, F(o_1)) \vee (\exists o_2 \in Cl_2, F(o_2)) \ldots \vee (\exists o_n \in Cl_n, F(o_n))$.

Checking strategy : Various strategies can be used to check a disjunction of local constraints of the form $C \equiv C_1 \vee C_2 \ldots \vee C_n$. It is necessary to determine, among the local constraints C_i, those that are touched by the leaves of the nested transaction. To simplify, suppose that those local constraints are $C_1, C_2 \cdots C_k$, with $k \leq n$ (in other words, we suppose that the constraints $C_{k+1}..C_n$ are not touched by the nested transaction).
A first method to check C is the following one: perform the checking of local constraints $C_{k+1} \cdots C_n$ in parallel with the execution of the transaction (i.e. without waiting for the end of the execution of the leaf transactions that touch the constraint). As these local constraints are not affected by the nested transaction, the satisfaction of one of them is enough to guarantee the satisfaction of the global constraint, whatever the effects of the transaction. In this case, it is not necessary to check the local constraints $C_1 \cdots C_k$ after the execution of the leaf transactions. If none of the local constraints $C_{k+1} \cdots C_n$ is satisfied, then $SCA(C)$ must wait for the end of the leaf transactions that touch C, to execute the local checking processes for $C_1 \cdots C_k$. This method has the drawback of accessing sites $(S_{k+1}, \ldots, S_n)$ not touched by the transaction. Its advantage is that it increases the parallelism, avoiding to wait for the end of the leaf transactions to start the checking process. Another possible method is to proceed conversely and to wait for the end of the transactions touching C to perform the checking. The checking is initiated on the related sites $(S_1, \ldots, S_k)$, and, if no local constraint is satisfied, $SCA(C)$ runs the checking process on $S_{k+1}, \ldots, S_n$ until finding a C_j that is satisfied. This method has the advantage to access sites not touched by the transaction only if necessary, but it reduces the parallelism. If leaf transactions touch constraints on few sites, a third method is to check C_j

before the execution of the leaf transaction on S_j . Indeed, if C_j is not satisfied, then it exists a site S_i, (different from S_j), *not touched by the transaction*, on which C_i is satisfied, since C, which is a disjunction, is supposed to be satisfied at the beginning of the transaction. In consequence, C will be satisfied, whatever the effects of the leaf transaction. If C_j is satisfied at the beginning of the transaction, then one of the two previous methods can be applied. Notice that these three methods can be optimized if the system maintains information about the satisfaction of the different components of the disjunction.

5.2 Global Constraints Non-locally Checkable

A constraint that can not be completely checked by independent local processes will contain quantified predicates that need a simultaneous checking on different sites. Such constraints might have an arbitrary complexity. They can be defined by using any predicate type, quantifier and connector, which makes impossible to establish a general checking strategy. In this paper, we focus on an important constraint class (as pointed out in the literature), which we call *conjunctive global constraints*. A constraint C of this class can be expressed as:

$$C : \forall(o_1 \in Cl_1, \cdots, o_n \in Cl_n)$$

$$F_1(\boldsymbol{x}_1) \ldots \wedge F_j(\boldsymbol{x}_j) \Rightarrow F_{j+1}(\boldsymbol{x}_{j+1}) \wedge F_{j+2}(\boldsymbol{x}_{j+2}) \ldots \wedge F_m(\boldsymbol{x}_m)$$

where F_i is any predicate and $\boldsymbol{x}_i$ is any subset of the variables $o_1, o_2, \cdots, o_n$. This type of constraints generally appears when chains of relationships exist between objects of different local databases. This is the case in the example of Figure 2 where a relationship exists between the class *Car* on site S_1 and the class *Person* on site S_2, through the reference *Owner*. On this multidatabase we can define a universal global constraint C_F :

$$C_F : \forall(v \in Car, p \in Person), (Mark(v) = Ferrari \wedge Owner(v) = p) \Rightarrow age(p) \geq 33$$

Constraint C_F states that the owner of a Ferrari must be at least 33 years old.

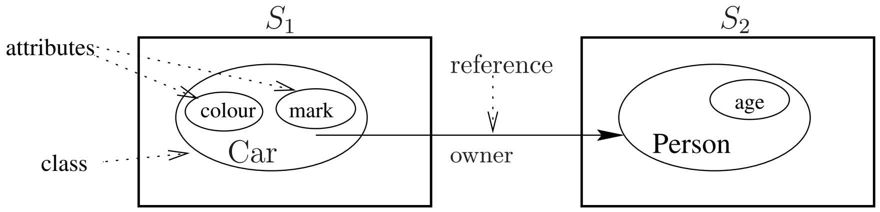


Fig. 2. Link between S_1 and S_2 through the Owner reference

As mentioned in Section 2, a classic optimization for the checking of a universally quantified constraint is to collect, at execution time, the objects modified by a transaction and to check the constraint only with regard to those objects (and

to those to which they are related through the constraint). To minimize the transfers of objects on which the constraint will be checked, we carry out an analysis of each constraint, with a double purpose.
The first purpose is inspired of [12]. We determine, for each site S_i and each global constraint C_j, where it is possible, a predicate locally checkable on that site. This predicate, designated from now on by $PL_{i,j}$, is such that if it is satisfied with regard to the modified objects, this guarantees the satisfaction of the global constraint. In the example of Figure 2, $PL_{1,F}$ corresponds to $Mark(v) \neq Ferrari$ and $PL_{2,F}$ corresponds to $age(p) \geq 33$. In the remainder of this section, we only consider one constraint at a time and thus we can omit the second indice in $PL_{i,j}$, i.e. $PL_{i,j} = PL_i$
The second purpose is based on the method proposed in [11]. We decompose a universally quantified global constraint into a conjunction of predicates through the definition of *inter-sites predicates*. The idea is to obtain a sub-constraint for each database which describes the responsibility of this database with regard to the satisfaction of the global constraint. We add a particular constraint that relates all the inter-sites predicates to the global constraint. This particular constraint is stored on a single site which is the one on which the global constraint will be finally checked. A universal global constraint can be decomposed in a number of ways equal to the number of sites involved by that constraint. For constraint C_F, there are therefore two possible decompositions :

1. If C_F is checked on S_1, C_F is re-written in the following way:
   On S_1**:**
   $\forall(v \in Car, p \in Person\_less\_33), Mark(v) = Ferrari \Rightarrow Owner(v) \neq p$
   On S_2**:** $\forall(p \in Person), age(p) < 33 \Leftrightarrow p \in Person\_less\_33$
   Person_less_33 is the set associated to the inter-site predicate. It contains the objects of the class *Person* touched by the transaction which do not satisfy the local predicate PL_2.
2. If C_F is checked on S_2, C_F is re-written in the following way:
   On S_1**:** $\forall(v \in Car, p \in Person\_with\_ferrari), (T(v) = Ferrari \wedge Owner(v) = p) \Leftrightarrow p \in Person\_with\_ferrari$
   On S_2**:** $\forall(p \in Person), p \in Person\_with\_ferrari \Rightarrow age(p) \geq 33$
   Person_with_ferrari is the set associated to the inter-site predicate. It contains identifiers of objects of the class *Person* touched by the transaction which do not satisfy the local predicate PL_1, i.e. the set of objects of the class *Person* which are owner of a *Ferrari*.

Checking Strategy : To check this kind of constraints, we keep all the possible decompositions and select, at checking time, the one which minimizes data transfers according to the sites where the transactions that touch the constraint are executed. The $SCA(C)$ launches, on its own site, the $Check(C)$ process which is in charge of checking C. This process, which is a sub-rtansaction of $SCA(C)$, is the root of a distributed nested transaction that will have a child in each site S_i where there exist leaf sub-transactions that touch C. Each child, called $Check(CMJ_i)$, has to check C with regard to updates made on S_i. It has the following behavior:

1. Check PL_i if it exists. If PL_i is satisfied, it will not be necessary to carry out other actions, since C will be satisfied with regard to the modifications made on S_i.
2. If PL_i is not satisfied or does not exist, the inter-sites predicates are used to identify the objects on S_i which are necessary to perform the checking of constraint C. Modifications carried out on S_i are sent to the site where C will be checked. Then the checking process begins.

Notice that this checking method originates a nested transaction $Check(C)$ distributed on different sites: the sites that contain leaf transactions that touch C and those where C should be checked. To illustrate this, reconsider constraint C_F defined on data of sites S_1 and S_2 in Figure 2. To check C_F, one among the three following transactions is generated, depending on the sites where leaf transactions that touch C_F are executed.

First Case: the leaf transactions that touch C_F are all executed on S_1. The structure of the transaction $Check(C_F)$ corresponds to the left sub-tree of the nested transaction in Figure 3. $Check(C_F)$ has a child $Check(CMJ_1)$ in charge of checking C_F with regard to the modifications carried out on S_1. $Check(CMJ_1)$ has a sequence (denoted by the symbol ';') of child sub-transactions. The first sub-transaction, $Check(PL_1)$, checks the local predicate PL_1 which, if satisfied, guarantees the satisfaction of C_F. If PL_1 is not satisfied, i.e. if the leaf transactions produce a Car with new value *Ferrari* for the attribute $Mark$, or if they assign a new *Owner* to a $Ferrari$, then it is necessary to build the set $Intersite_1$. This set will contain the objects p_j which are owners of those vehicles. This set will be sent to S_2 where $Check(CL_2)$ will check if all the owners p_j of $Intersite_1$ have an age greater than or equal to 33.

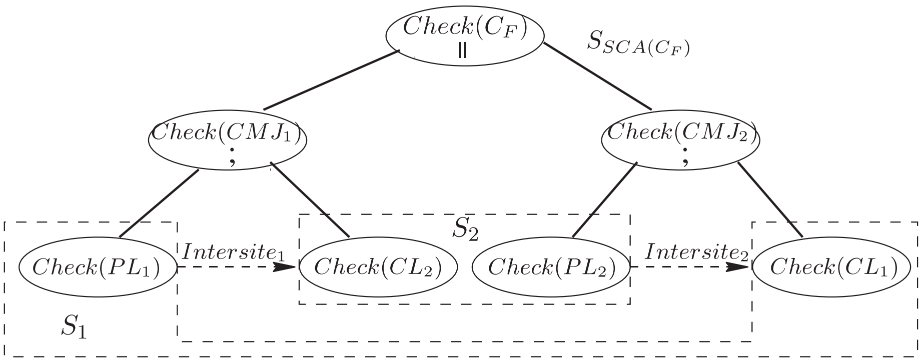


Fig. 3. Global integrity constraint touched by sub-transactions on both sites

Second Case: all leaf transactions that touch C_F are executed on S_2. This case is symmetrical to the previous one and corresponds to the right sub-tree of the nested transaction in Figure 3. Here the modifications concern the attribute *age(p)* of the objects of the class *Person*. If there are updates producing an *age* smaller than 33 for a *Person*, it is necessary to build the set $Intersite_2$

containing the objects p_j representing those people. This set is sent to site S_1 where we check if all the objects it contains are not owner of a $Ferrari$.
Third Case: the leaf transactions that touch C_F are executed on S_1 and S_2. A leaf transaction is executed completely on a single site, therefore there exist leaf transactions that touch the class *Car* and other leaf transactions that touch the class *Person*. Thus, modified objects must be sent from S_1 to S_2 and vice-versa. Figure 3 shows the structure of the transaction $Check(C_F)$. It has two subtransactions, $Check(CMJ_1)$ and $Check(CMJ_2)$, which are executed in parallel (denoted by the symbol $\|$). $Check(CMJ_i)$ checks constraint C_F with regard to all the leaf transactions that are executed on S_i. The process $Check(CMJ_1)$ and the process $Check(CMJ_2)$ use each one a different set of objects, $Intersite_2$ and $Intersite_1$ respectively, which allows a parallel execution.

6 Conclusion

In this paper, we have studied different checking strategies for integrity constraints defined in a declarative way in a multidatabase environment supporting nested transactions. As in [8], the control of the checking process of a constraint is performed by the smallest common ancestor of the leaf sub-transactions of a nested transaction touching the constraint. This approach allows to guarantee the consistency of the multidatabase even in the presence of global constraints, (i.e. constraints involving data located on several sites) while reducing the number of sub-transactions to abort, in case of partial abortion of the nested transaction. In order to adapt the solution of [8] to a multidatabase environment, we show how to minimize the checking cost by minimizing the transfer of data among sites. We propose a classification of strategies depending on the constraint nature and on the structure of the transactions. Through examples we illustrate how, in some cases, constraint checking is performed as far as possible by processes using only local data. These processes may themselves be part of a nested transaction, allowing to parallelize the checking process. The main classes of global constraints we consider are those expressed by a conjunction (disjunction) of local constraints, and an important class of global constraints universally quantified. For the latter, we use a method of constraints decomposition described in [11] as well as the definition of predicates locally valuables that guarantee the satisfaction of global constraints [12]. Our main contribution with respect to those constraints is to integrate the approaches of [11] and [12] into a framework which takes advantage of nested transactions for an early and efficient checking of constraints.
As far as we know, our approach is the only one dealing with global constraint checking for distributed nested transactions. Other approaches for distributed integrity constraints checking [10,11] only consider distributed transactions models with a single nesting level.
As short-term work, we plan to implement this method on the medical images management system SIMA [14]. As future work, we want to adapt our approach to other transactions models (multi-level transactions, sagas, open-nested trans-

actions, etc). We also plan to test and validate our system on applications, such as e-business.

References

1. Transarc Encina Product Information. http://www.transarc.com/Solutions. http://www.transarc.com/Product/Txseries/Encina/. 317, 318
2. V. Benzaken and A. Doucet. Thémis: A Database Programming Language Handling Integrity Constraints. *The VLDB Journal*, 4(3):493–517, July 1995. 317
3. J. Besancenot, M. Cart, J. Ferrié, R. Gerraoui, P. Pucheral, and B. Traverson. *Les systèmes transactionnnels*. Hermes, Paris, 1997. 317
4. B. T. Blaustein. *Enforcing Database Assertions*. PhD thesis, Harvard University, Cambridge, MA, 1981. 317
5. E. Boertjes, P. W. P. J. Grefen, J. Vonk, and P. M. G. Apers. An Architecture for Nested Transactions Support on Standard Database Systems. In G. Quirchmayr, E. Schweighofer, and T. J. M. Bench-Capon, editors, *Proc. 9th Int. Conf. Database and Expert Systems Applications, DEXA'98*, volume 1460 of *LNCS*, pages 448–459, Vienna (Austria), August 1998. Springer-Verlag. 317, 318
6. S. Conrad, M. Höding, S. Janssen, G. Saake, I. Schmitt, and C. Türker. Integrity Constraints in Federated Database Design. Technical Report 2, Fakultät fur Informatik, Universität Magdeburg, April 1996. 316
7. B. Defude and H. Martin. Integrity checking for Nested Transactions. In R. Wagner and H. Thoma, editors, *Proc. 7th Int. Conf. Database and Expert Systems Applications, DEXA'96*, pages 147–152, Zurich (Switzerland), September 1996. IEEE-CS Press. 317
8. A. Doucet, S. Gançarski, C. León, and M. Rukoz. Nested Transactions with Integrity Constraints. In G. Saake, K. Schwarz, and C. Türker, editors, *TDD'99, Dagstuhl Castle, Germany, September 27-30, 1999, Selected Papers*, volume 1773 of *LNCS*, pages 130–149, Berlin, 2000. Springer-Verlag. 316, 317, 318, 326
9. P. W. P. J. Grefen and P. M. G. Apers. Integrity Control in Relational Database Systems — An Overview. *Data & Knowledge Engineering*, 10:187–223, 1993. 316, 317
10. P. W. P. J. Grefen and J. Widom. Protocols for Integrity Constraint Checking in Federated Databases. *Distributed and Parallel Databases*, 5(4):327–355, October 1997. 326
11. S. Grufman, F. Samson, S. M. Embury, P. M. D. Gray, and T. Risch. Distributing Semantic Constraints Between Heterogeneous Databases. In Alex Gray and Per-Åke Larson, editors, *Proceedings of the Thirteenth International Conference on Data Engineering, ICDE 1997, April 7-11*, pages 33–42, Birmingham U. K., April 1997. IEEE Computer Society. 316, 324, 326
12. A. Gupta and J. Widom. Local Verification of Global Integrity Constraints in Distributed Databases. In P. Buneman and S. Jajodia, editors, *Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data*, volume 22 of *ACM SIGMOD Record*, pages 49–58, Washington, D. C.(USA), May 1993. ACM Press. 316, 324, 326
13. J. E. B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing*. MIT Press, Cambridge, MA, 1985. 317, 319
14. M. Rukoz, C. León, and M. Rı́vas. SIMA: A Java Tool for Constructing Image Processing Applications on a Heterogeneous Network. *to appear in Parallel and*

Distributed Computing Practices. Special Issue on Distributed Object Systems. 317, 318, 326

15. A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990. 319
16. A. P. Sheth, M. Rusinkiewicz, and G. Karabatis. Using Polytransactions to Manage Interdependent Data. In A. K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, pages 555–581. Morgan Kaufmann Publishers, San Mateo, CA, 1992. 317
17. G. Weikum, A. Deacon, W. Schaad, and H.-J. Schek. Open Nested Transaction in Federated Database Systems. *IEEE Data Engineering Bulletin*, 16(2):4–7, June 1993. 319

The Internet Marketplace Template: An Architecture Template for Inter-enterprise Information Systems[⋆]

Mark A. Cameron, Kerry L. Taylor, and David J. Abel

Enterprise Distributed Systems Technology CRC and
CSIRO Mathematical and Information Sciences
GPO Box 664 Canberra, 2601, Australia
{mark.cameron,kerry.taylor,dave.abel}@cmis.csiro.au

Abstract. Modern enterprise inter-connectivity should also enable a market for third-party value-added information resellers. Resellers can offer integrated information products, made out of information sourced dynamically from multiple heterogeneous enterprise information resources. The Internet Marketplace is an information infrastructure designed for this environment. In this paper we address the design of inter-enterprise information systems by introducing a UML architectural template for services and applications built on the Internet Marketplace information infrastructure. The template is applied to a particular case study, involving a federation of state and local government agencies. The Internet Marketplace Template addresses some of the unique constraints of this type of application, offering application designers a tool for modelling and designing inter-enterprise applications that make use of the information infrastructure.

1 Introduction

The wide accessibility of corporate information stores, made possible by the pervasiveness of the Internet, enables a new generation of inter-enterprise applications. Enterprises are responding to new business imperatives by entering into strategic alliances that require the routine exchange of operational documents and data. Although sometimes brokered, these are primarily peer-to-peer, one-on-one transactions that replicate pre-e-commerce business models. They are aimed at business-to-business or business-to-consumer information exchange [14]. For the most part, the information exchanged is essentially meta-data, used to support the order, payment and delivery processing of a material product such as contracted service delivery, securities, books, machinery parts, and so on. Sophisticated marketplaces are emerging, to support tasks such as supply chain management (e.g. www.sapmarkets.com, www.rosettanet.org, [2]), that also promote information exchange among enterprises.

⋆ This work supported through the Commonwealth Government's Cooperative Research Centre for Enterprise Distributed Systems Technology.

© Springer-Verlag Berlin Heidelberg 2001

Some enterprises have taken advantage of the Web to offer digital information services, whereby the product offered is itself in digital form. This approach is well developed for digital music (for example, www.mp3.com), industry analysis reports (for example, www.idc.com), and more complex earth observation data, for example [10]. Nevertheless, greater opportunities exist for the provision of virtual information products that are custom-made out of enterprise information resources. The potential for trade in demand-driven integrated inter-enterprise information products has been largely ignored. Some government enterprises have legal obligations, policy drivers or stakeholder demands to open their information resources to public access. They may even encourage the development of an industry based on value-added information products using their information resources. Value-added products could be embodied as inter-enterprise applications that are concerned with user interaction, information presentation and some integration logic.

In a marketplace environment for data, where data extracted from individual data sources can be repackaged, reprocessed, and re-presented in an integrated form, it is only a small step to expand to include the participation of data processing resources as well [4]. In this case, vendors of analytical algorithms could enable their algorithms, embedded in software code, to be used in the marketplace along with the data resources.

In this paper we are concerned with both the design of an infrastructure for the enablement of digital Internet marketplaces and an indicative application of this infrastructure applied to spatial information sharing. In our terminology, an *Internet Marketplace* (IMP) is a community of *request service providers* and infrastructure services that support the synergetic combination of services to produce value-added products and services. Request services are *query services* or *function services*. Value added services might themselves be offered as request services or as end user applications.

We present an architectural template for both marketplace request services and applications in the Unified Modeling Language, UML [13]. It is an architecture because it describes the organizational structure and behaviour of the system. It is a template because it describes only those aspects of the system that are application domain-independent and focuses on the way an inter-enterprise application or value-added service works in the marketplace framework. We describe aspects of each of the Use case view, the Design view (through interfaces and example bindings) and the Deployment view. In this paper we leave aside the Implementation view and the Process view (which is very simple in the marketplace). The views describe only those aspects of the infrastructure that are necessary to explain the relationship between applications and request services. We also apply the template to a particular marketplace application, refine it for that purpose and consequently offer a Deployment view. As a whole, the template describes the design and function of a marketplace value-added service and a marketplace application within the IMP framework. This encourages the development of third-party value-added services and applications in the marketplace.

Section 2 provides an outline of the IMP infrastructure and component services, which are described in [1], as they relate to inter-enterprise applications. Section 3 is an overview of the IMP Architecture Template. This template is the basic framework for documenting marketplace designs using the IMP infrastructure. Section 4 provides a case study of the IMP Architecture Template applied to a problem in state and local government in New South Wales (NSW), Australia. Section 5 places the IMP Infrastructure and the IMP Architecture Template in context with other integration and design frameworks. We present our conclusions and scope for future work in Sect. 6.

2 The Internet Marketplace

Our approach to enabling inter-enterprise applications is based on developing a re-usable information infrastructure. One approach to encourage infrastructure reusability is through declarative specifications. Specifications should enable dynamic realisation of inter-enterprise application data from the data and functions of the existing *application*, *domain* and *persistence* layers of an enterprise information system architecture. A specification-based infrastructure provides inter-enterprise application developers with a higher level of abstraction than an infrastructure based on bespoke interface adapters. The intention of our declarative specification approach is to require that applications (dynamically or statically) formulate and present to the infrastructure a specification of what the application requires from a service in terms of an externally visible domain model, constraints against that model, and finally, the restructuring and encoding required by the application. This focus on specification moves the engineering problem away from the seam between enterprise systems and inter-enterprise applications and onto construction (at the inter-enterprise application end) and interpretation (at the enterprise end) of the specification.

Services establish their own externally visible domain model by registering an external schema in the registry service, which is part of the IMP inter-enterprise infrastructure. This registry service provides details of the external schemas, structural transformation options, output presentation formats and capabilities of request services participating in the inter-enterprise information infrastructure. The registry service fulfils three important roles: Type Registry, Semantic Registry and Catalogue. The role of a Type Registry within the solution context is to document the sets of data types and their operators and functions that a service provider can draw on to describe the capabilities of a request service wrapper. We assume that the service providers have established these collectively within a marketplace. Clearly then, the Type Registry has a critical role in establishing a minimum set of abstract data types useful to a community of request services. The Semantic Registry role provides the essential ontology for the community of interest. The role of a Catalogue within the solution context is to present to customers and inter-enterprise applications, the collection of services participating in the marketplace, together with locations and protocols, external schemas and capabilities of those services. The capabilities of a service are expressed in terms of the operators and functions of marketplace-specific data types that a service implements. The registry contains a representation, service by service, of each domain model exported by each service. These external schemas represent starting models offered by services.

Applications specify their requirements in terms of a declarative request targeted at a specific service. In this way, application interaction with the infrastructure becomes specification driven. Our model of a declarative request allows optional constraints against the external schema of the service and an output clause, which specifies the encoding and possible restructuring required by the application. Restructuring enables an application with a different structural view of a service domain model to specify how the service should map its existing domain model onto the view required by an application. Encoding enables applications and services to reduce the request complexity by using an agreed data transfer format.

3 Internet Marketplace Template

The enterprise systems, together with their request services, external schemas, data, exchange formats, declarative request language and capabilities form the IMP inter-enterprise information system infrastructure. We now present the Internet Marketplace Template, which is based on the IMP inter-enterprise information system infrastructure and follows the guidelines above.

3.1 Use Case View

We first describe the key *actors* and *use cases*, shown in Fig. 1, delineating the environment and actor requirements addressed by the template.

An *Enterprise* is an autonomous operating entity. For our purposes, an *Enterprise System* represents those automated software systems and data that an enterprise uses and generates in conducting its business. An enterprise voluntarily participates in the inter-enterprise information system infrastructure by exposing data, processing and/or enterprise systems through one or more *Request Services.*

Customers use an *Inter-enterprise Application* to *satisfy customer requirements.* Customer requirements are strictly abstract within the template, since they are outside the template's scope. When using the template, inter-enterprise application builders will instantiate and refine these requirements together with the application.

A *Request Service* provides a declarative request seam (or interface) to inter-enterprise applications. When *handling declarative requests*, each request service is responsible for mapping a declarative request, against the service's external schema, into queries and actions against the target enterprise system and transforming the result into the structure and format requested.

Value added reseller services (*VAR Request Service*) and request services are indistinguishable to customers and applications. A *VAR Request Service* will also handle a declarative request, but this type of service is different to a *Request Service* in that the *VAR Request Service* will potentially reuse data and or data processing functions provided by other inter-enterprise request services in fulfilling its declarative request. Typically a *VAR Request Service* will need to transform its request requirements into one or more declarative requests for other inter-enterprise request services, perhaps using local data to do this.

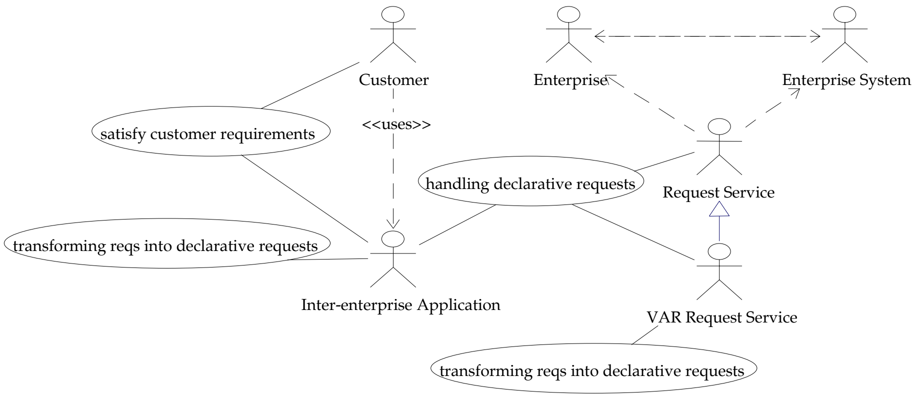

Customer
Enterprise
Enterprise System
<<uses>>
satisfy customer requirements
handling declarative requests
Request Service
transforming reqs into declarative requests
Inter-enterprise Application
VAR Request Service
transforming reqs into declarative requests

Fig. 1. Requirements for Customer, Enterprise, Inter-enterprise Application, VAR Request Service and Request Service

An *Inter-enterprise Application* will *satisfy customer requirements*, in part, by *transforming requirements into declarative requests* and, together with a number of target enterprise system request services, *handling declarative requests*. Parts of the process of transforming customer requirements into declarative statements are outside the scope of the template, however every inter-enterprise application operating within the infrastructure will perform this process. We have observed three variants of the transformation process, namely: static, parameterised and dynamic request generation, shown in Fig. 2. When using the template, inter-enterprise application builders instantiate and refine the transformation process using one or more of these basic request generation approaches. *Application Developers* use the *Inter-enterprise Registry Service* to assist in specifying requests.

Developers may choose *static request generation* and embed static requests into their application where there is a direct mapping from customer and application or VAR request service requirements onto the inter-enterprise infrastructure environment. That is, the application requests information, perhaps also with structural transformations, in a known presentation format, from a single request service's supported external schema. Typically the application will manage and manipulate the data directly. We have used this approach in web pages

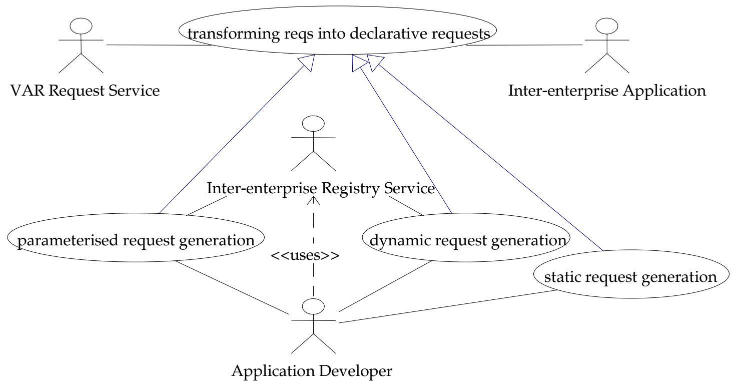

transforming reqs into declarative requests
VAR Request Service
Inter-enterprise Application
Inter-enterprise Registry Service
parameterised request generation
<<uses>>
dynamic request generation
static request generation
Application Developer

Fig. 2. Transformation Variants

with static declarative requests for data, typically in JPEG, XML and HTML formats, from request services.

Other applications or VAR request services address their requirements through *parameterised request generation* of a declarative request. In the simplest case, the developer will construct a declarative request template with parameterised constraints, providing constraint values at runtime. We have used this technique for applications and VAR request services where the service addressed by the request and the data format returned as a result of the request does not change, allowing a high degree of certainty that the request will return an answer.

More complex parameterisations are possible, ultimately leading to *dynamic request generation*. We envisage that query planning services (where the service offered is to translate a domain specific request into a plan consisting of a series of requests against other VAR and request services) would realise the dynamic request generation process. In any case, these dynamically generated requests need reconciliation with the capabilities of request services.

3.2 Design View

As Fig. 3 shows, the seam between the inter-enterprise application and the information infrastructure is programmatically simple, though it does depend upon an agreed communication protocol. An information infrastructure must accommodate a number of potential communication options. We have experimented with CORBA and HTTP bindings for the declarative request service interface, though other bindings, such as SOAP, are possible. These bindings simply provide a transport layer for the declarative request and associated response. The template does not specifically address the implementation issues of designing and building request service wrappers.

Request service has two specializations: *Query Service* and *Function Service*. Query services are query only services that accept declarative query requests,

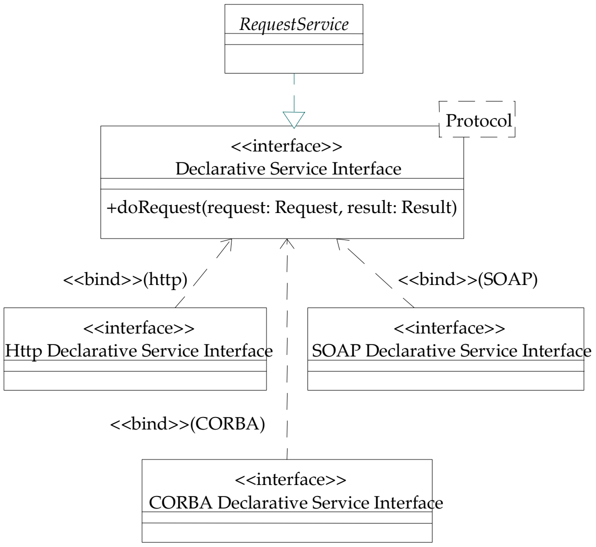

RequestService
Protocol
<<interface>>
Declarative Service Interface
+doRequest(request: Request, result: Result)
<<bind>>(http)
<<bind>>(SOAP)
<<interface>>
Http Declarative Service Interface
<<interface>>
SOAP Declarative Service Interface
<<bind>>(CORBA)
<<interface>>
CORBA Declarative Service Interface

Fig. 3. Interface Bindings

and return data satisfying the request in the structure and format specified by the request. Function services supply data, derived from information provided on invocation of the service, possibly with use of data from a data store private to the service.

The inter-enterprise registry service documents the structure of information products offered by the infrastructure. This defines the initial domain models offered by request services. The request specification language and the inter-enterprise registry service use a non-first-normal-form model for schemas. The template uses stereotypes to assist request service designers in modelling the external schema of their service. The stereotypes capture the extent (<<service class>>, <<service variable>>), intentional structure (<<service constructed type>>) and functional navigation abstractions (<<service class association>>) of a request service schema, as well as modelling the primitive (<<marketplace primitive type>>) and constructed (<<marketplace constructed type>>) intentional structure of agreed marketplace types, operators (<<marketplace operator>>) and functions (<<marketplace function>>). Figure 4 is an example model of marketplace types and some request service classes. For brevity we have shown only some of the stereotypes and we will explain the example more fully in sect. 4.1.

The registry service is a specialisation of a *Query Service*. The registry service is like any other request service participating in the infrastructure, in that it offers an external schema and implements our declarative request specification language. VAR request services and applications may query and use information contained in the registry service, such as identifying services offering schemas containing data of a specific marketplace type, when fulfilling their runtime obligations.

Figure 5 is a schematic of the IMP infrastructure components. Together, the declarative request specification language, the marketplace types, operators and functions, the declarative service interface and the implementing request service wrappers, their external schemas and the inter-enterprise registry service form an inter-enterprise information system infrastructure.

We now present the Sydney Information Highway case study, showing how we used the IMP Infrastructure and Template to design and deploy an inter-enterprise application.

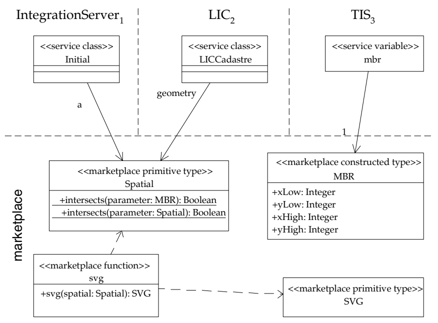

IntegrationServer$_1$
LIC$_2$
TIS$_3$
<<service class>> Initial
<<service class>> LICCadastre
<<service variable>> mbr
a
geometry
1
<<marketplace primitive type>> Spatial
+intersects(parameter: MBR): Boolean
+intersects(parameter: Spatial): Boolean
<<marketplace constructed type>> MBR
+xLow: Integer
+yLow: Integer
+xHigh: Integer
+yHigh: Integer
marketplace
<<marketplace function>> svg
+svg(spatial: Spatial): SVG
<<marketplace primitive type>> SVG

Fig. 4. IMP Inter-Enterprise Registry Service Content

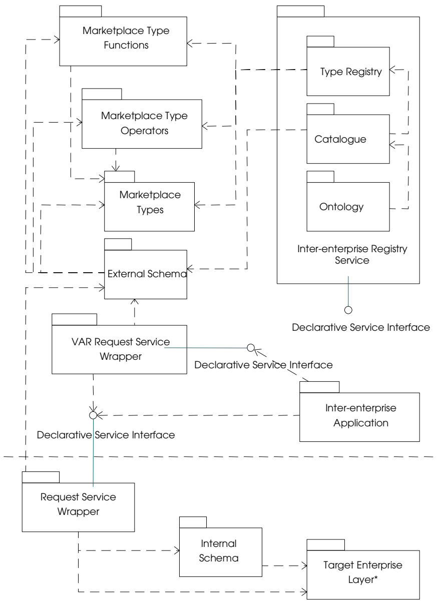

Marketplace Type Functions
Marketplace Type Operators
Marketplace Types
External Schema
Type Registry
Catalogue
Ontology
Inter-enterprise Registry Service
Declarative Service Interface
VAR Request Service Wrapper
Declarative Service Interface
Inter-enterprise Application
Declarative Service Interface
Request Service Wrapper
Internal Schema
Target Enterprise Layer*

Fig. 5. Internet Marketplace Inter-enterprise Information System Infrastructure

4 The Sydney Information Highway Case Study

In Australia, state governments have historically had responsibility for land use decision-making. Within the states, regional and local government councils are responsible for introducing planning schemes. These schemes indicate what land use is permitted or restricted, under what conditions, and over which areas (zones). When planning schemes at state, regional or local levels conflict, state schemes prevail over regional schemes, which prevail over local schemes. The tasks of assembling (by property developers) and assessing (by agency officers) development proposals requires navigating state and local government planning documents and is complicated by the fact that zoning schemes vary across council boundaries.

The Inner Metropolitan Regional Organization of Councils (IMROC) is a voluntary body of eight local government councils[1] within Sydney. IMROC initiated a partnership agreement for the revitalisation of Parramatta Road with New South Wales state government departments[2], agencies[3], businesses, the community and other local councils[4] with jurisdiction for Parramatta Road. The goals of the revitalisation project as they relate to this case study are to broadly improve coordinated land planning by providing land use planning and other spatially referenced information to state and local government officers, businesses and the wider community, via the Internet. A significant project constraint was to achieve these objectives without constraining the decisions of partner agencies with respect to the implementation and development of their own property and management information systems, servers and Geographic Information Systems (GIS) environment.

There is a significant amount of data duplication, both across state and local government tiers, and among state government agencies. Update cycles for duplicated data vary, and often require extensive transformations and conversions when moving across state and local government tiers. Councils would like to speedup information dissemination and believe that Internet delivery mechanisms provide an opportunity to achieve this aim. Searching for properties based on street address is functionality that requires improved access to information across government tiers.

4.1 Template Application to Sydney's Information Highway

IMROC, in consultation with partnership stakeholders, developed a requirements model for their inter-enterprise application, consisting of a *Casual User*, a *High Level User*, and an extensive set of functional requirements. For brevity of presentation, we summarise the actor and stakeholder requirements as: interactive display of spatial data; on-demand query and display of linked zoning and planning documents; on-demand display of spatial meta-data; spatial data download using an open spatial data exchange format; street address query; and use the existing information systems and spatial data infrastructure of stakeholders, where possible.

We assigned the *Casual User* and *High Level User* requirements to the SIH application layer, an instantiation of the *Inter-enterprise Application* and *satisfy*

[1] Ashfield, Burwood, Concorde, Drummoyne, Lane Cove, Leichhardt, Strathfield and City of Sydney.

[2] Department of Information Technology and Management (DITM), Department of Land and Water Conservation (DLWC), Department of Public Works and Services (DPWS) and Department of Urban Affairs and Planning (DUAP).

[3] Land Information Centre (LIC), Roads and Traffic Authority (RTA).

[4] Auburn, Holroyd, Marrickville, South Sydney.

customer requirements. The application layer consists of a presentation applet and an integration server. Java's default security measures prevent applets from requesting connections to other hosts, primarily motivating the client-server split of the application layer in this way. We built the applet using an early version of the CSIRO SVG Toolkit[5], an open-source Scalable Vector Graphics (SVG) [6] document renderer. The SVG Toolkit enables hypermedia linking of vector spatial entities; a feature we use to direct declarative requests, embedded in the SVG document, to the request service wrappers of the inter-enterprise infrastructure.

We fitted a request service wrapper to the integration server, enabling the server to act as a VAR request service with its own external schema. The applet generates and submits parameterised declarative requests to the integration server. All of the applet's generated requests specify one of the spatial data formats (SVG, compressed SVG, or MIFMID) as the return format. A typical request string, using the HTTP protocol, is:

```
http://imp.cmis.csiro.au/IMROCClientServlet?RSL=#InitialData
intersects mbr(313162,1248539,313445,1248737)#svg(InitialData)
```

The integration server's external schema is logically a number of views over the external schemas of the participating request service wrappers. In the example request (and with reference to Fig. 4), `InitialData` is logically composed of data from the schemas of council boundaries, cadastre, Parramatta Road and a backdrop image, sourced from different request service wrappers. One attribute of the `InitialData` class is of `spatial` type, one of the primitive marketplace types supported by the request service wrappers and recorded in the registry. The `spatial` type has a marketplace type operator `intersects`, which takes a constructed marketplace type `mbr`. The example contains a literal representation of that type. The marketplace type function `svg` operates on any class containing an attribute of type `spatial`, rendering its structure as an `SVG` document. Thus, the example demonstrates the performance of an operator (`intersects`) on two data types (`spatial`, `mbr`) and the encoding of the result using a function (`svg`).

The integration server is primarily responsible for hiding the data distribution problem from the applet. It does this with reference to the registry by re-writing the applet's request into a parameterised task tree. Each leaf task is a query against a relevant request service. The server executes the task tree (in parallel where appropriate) combining the document data streams into a single result stream and finally returning the result.

Stakeholder requirements were assigned to request service wrappers. An analysis of the existing infrastructure and council requirements identified four distinct request service candidates (apart from the integration server): MapXWrapper, ShapefileWrapper, RDBMSWrapper and TISWrapper, shown in Fig. 6.

The request services meet requirements at the *persistence*, *domain* and *application* layers of the existing enterprise architectures as follows:

persistence layer City of Sydney wished simply to provide access to spatial and attribute data using the ESRI shapefile format [5]. We built a shape-file

[5] Available at `http://sis.cmis.csiro.au/svg/`.

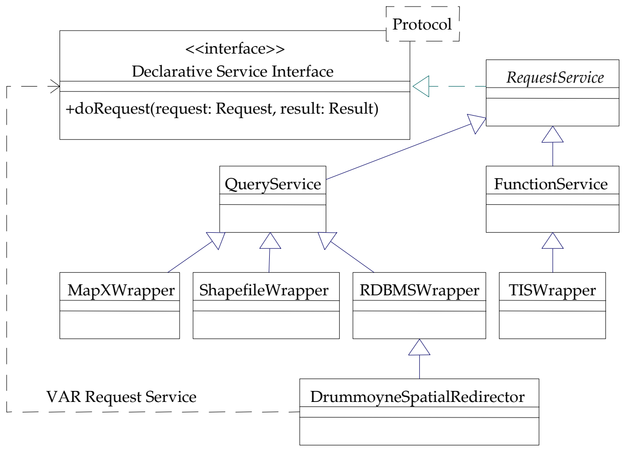


Fig. 6. Request Service Wrappers

wrapper supporting the declarative interface for shape-files. This wrapper needed to provide constraint testing against the shape-file in order to select the entities of interest as well as formatting operations to transform the data into XML, HTML, SVG and MIFMID. Other councils preferred to store and expose their spatial, property and address data through relational databases. The database wrapper transforms RSL requests into SQL statements, executes the statements, transforms and formats the data according to the request. The RTA, Ashfield, Leichhardt and Drummoyne council data use the database wrapper.

domain layer Some councils and agencies have MapInfo products for managing their mapping requirements. MapInfo provides access to a rich object-oriented domain model through their MapX product [8]. The MapX wrapper is targeted at this domain model and provides the same transformations and presentation formats as the persistence layer wrappers. MapX wrappers serve data for LIC, Ashfield, Marrickville, Holroyd and Parramatta councils.

application layer LIC has made high quality aerial photography of the region available for use by the application. We exposed the Tiled Image Server, an image application, to the infrastructure through a simple request service wrapper. This application stores image data in tiles and will re-assemble it on request. The wrapper provides only two export formats, SVG and JPEG, though the application supports many more.

Finally, though Drummoyne Council does not hold digital cadastre, it is able to offer a digital cadastre, sourced from LIC and augmented by Drummoyne's street address data, through Drummoyne's VAR service wrapper. The `DrummoyneSpatialRedirector` in Fig. 6 is another example of a VAR request service. This wrapper provides an external schema consisting of street address data and a cadastre for properties within its boundary. The Drummoyne wrapper manages requests for street address data by reference to its local database.

Drummoyne implements the virtual cadastre by forwarding requests for digital cadastre to the LIC (MapX wrapper) service. Drummoyne handles requests for a property boundary given a street address by mapping the address to a cadastre identifier, generating a declarative request for that cadastre and then forwarding the generated request to the LIC service.

4.2 Deployment View

For brevity of presentation, Fig. 7 shows both components of the information infrastructure and a partial deployment configuration for the wrappers. Figure 8 is an image of the user interface applet for Sydney's Information Highway. The applet presents users with a spatial display and interaction interface as well as a layer display and control interface. Casual users may refresh the document. This process, handled by the integration server, will discover additional spatial data themes (i.e. those themes that are not part of the default set) relevant to the current view. The system is available for public access at `http://imp.cmis.csiro.au/imroc/csiro`.

The project proceeded on a staged delivery schedule, requiring infrastructure development and deployment effort of approximately 10 man-months in parallel with application (user-interface and integration server) development effort of approximately 8 man-months, delivered in 3 months elapsed time. In addition to this effort, an IMROC officer provided contact coordination for the councils and agencies.

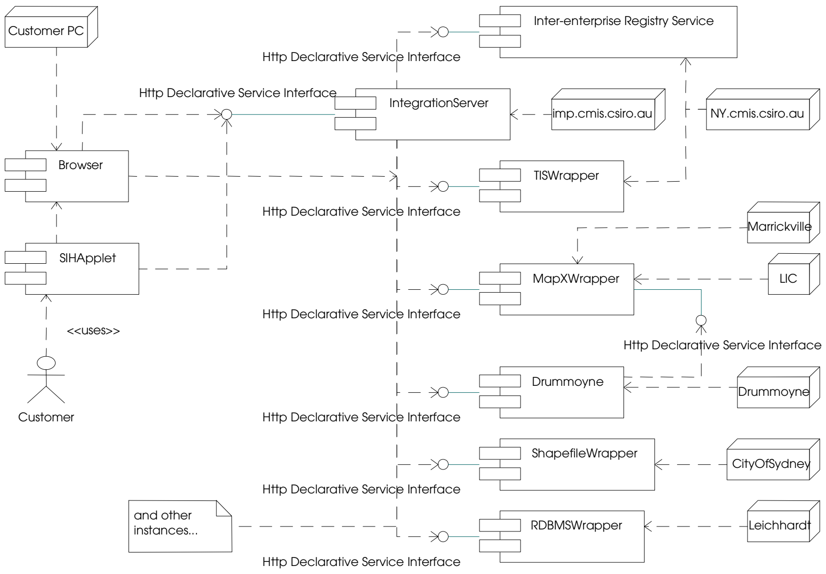
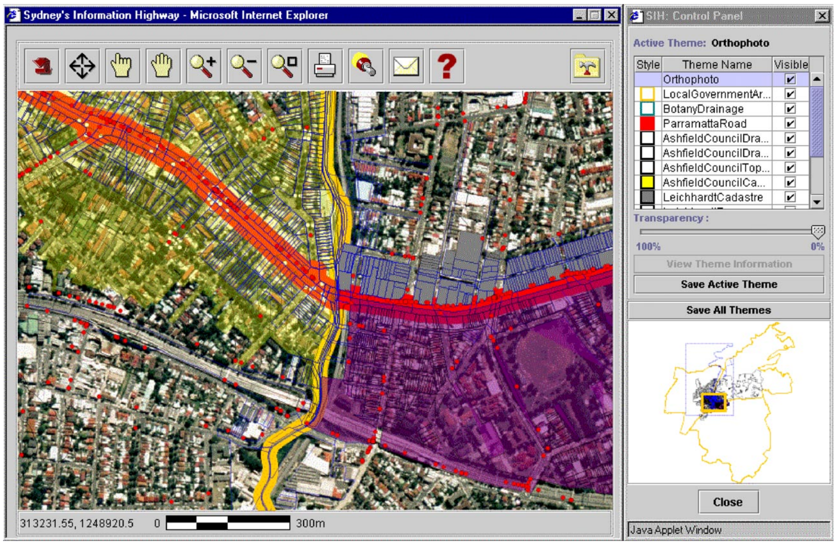

Customer PC
Inter-enterprise Registry Service
Http Declarative Service Interface
Http Declarative Service Interface
IntegrationServer
imp.cmis.csiro.au
NY.cmis.csiro.au
Browser
TISWrapper
Http Declarative Service Interface
Marrickville
SIHApplet
MapXWrapper
LIC
Http Declarative Service Interface
<<uses>>
Http Declarative Service Interface
Customer
Drummoyne
Drummoyne
Http Declarative Service Interface
ShapefileWrapper
CityOfSydney
Http Declarative Service Interface
and other instances...
RDBMSWrapper
Leichhardt
Http Declarative Service Interface

Fig. 7. SIH Inter-enterprise Information System Infrastructure



Active Theme: Orthophoto

Style	Theme Name	Visible
	Orthophoto	✓
🟨	LocalGovernmentAr...	✓
🟦	BotanyDrainage	✓
🟥	ParramattaRoad	✓
⬜	AshfieldCouncilDra...	✓
⬜	AshfieldCouncilDra...	✓
⬜	AshfieldCouncilTop...	✓
🟨	AshfieldCouncilCa...	✓
▨	LeichhardtCadastre	✓

Fig. 8. Sydney's Information Highway

5 Related Work

Solutions to the problems of interoperability and legacy system integration are crucial to internal enterprise business systems, in addition to inter-enterprise systems. Many technological solutions are available (which is not to say that all the problems are solved!) [3]. The IMP infrastructure approach is best suited to environments where there is a very loose relationship between data providers and data consumers, and where there is a will to promote end-user focused value-added applications by third parties. Sometimes these conditions arise within enterprises where there are very weak connections between business units, often the result of mergers or acquisitions. In an organization structured around individual or small group initiative and reward, there may not be a clear driving force to "standardise" across the organization, yet there may be pockets of awareness within business units that recognise opportunities to improve individual goals if distant corporate information were made available. This kind of environment is then quite similar to the inter-enterprise marketplace environment, and the IMP approach to enterprise integration will apply.

In this paper we have presented our architecture template in terms of UML and views of a system's architecture, in particular: use case view; design view and deployment view. The ODP Reference model [11] is an alternative framework we could use to describe a distributed software system's structure and behaviour. In terms of the ODP model we are directed at the information viewpoint, because we see the marketplace as primarily an information-providing service and we have focused on this aspect. Elements of the computational and engineering viewpoints are also presented, as these are necessary to meet our aims of describing how a value added service or application can be constructed. We have referred to the technology viewpoint only for explanation of the realisation in the case study. We have not explicitly addressed the Enterprise viewpoint, more relevant to the marketplace at large, in this paper. In any case, our computational and engineering solutions are formulated for an environment where the Enterprise viewpoint (in our case enterprises contributing to a marketplace) is only weakly specified, in a loosely collaborative environment.

Middleware for Method Management (MMM) [7] is a computational service infrastructure for deployment and use of distributed application services on the web. The MMM infrastructure defines an object model for infrastructure entities consisting of data set objects (DSO), method service objects (MSO) and method plan objects (MPO). The MMM infrastructure uses XML-DTD representations of the object model and XML documents to define object instances. There is a clear contrast here between how MMM exposes its infrastructure and how the IMP model, through a declarative request service interface and declarative request specification, exposes its infrastructure.

AeroWEB [9] uses the Systems Integration Architecture (SIA) as the basis of an information infrastructure that integrates information entities from a limited set of datasets, functions, applications, Functional Transformation Agents (FTA), users and projects. AeroWEB builds upon the technical architecture of CORBA to realize the SIA. AeroWEB achieves collaboration by enabling users to send packages of information entity references to others, who are then able to launch the application associated with the dataset and have the application interface appear on their screen. Issues of dataset transformation, where different applications use the same underlying data but require the data be restructured or reformatted, were not addressed in the AeroWEB project. SIA requires a launcher or dispatcher server on each machine offering a dataset or an application. These servers provide infrastructure access to the dataset or application required. The IMP infrastructure and the Internet Marketplace Template use wrappers to provide access to data or data processing services.

The IMP infrastructure is different to SIA in that it does not directly target collaboration; rather it addresses the issues of data and function interoperability by requiring users of the infrastructure (and these are typically though not necessarily applications) to declaratively specify data constraints, schema transformations and presentation formats required from a targeted request service. IMP data processing services are different to the application launching offered by AeroWEB. The IMP infrastructure focuses on data interoperability rather than universal application access.

The interoperability goals of the IMP infrastructure are quite similar to those referenced in Cooperative Information Systems (for example, [12]). The IMP infrastructure differs in implementation detail. The task of establishing a base of common knowledge for interoperability is placed on the representations of data, services and their capabilities, held by the registry, rather than devolving that responsibility to intelligent information agents.

6 Conclusions

In this paper we have outlined the objectives of an Internet Marketplace infrastructure for inter-enterprise applications. Based on those objectives, we have developed architectural descriptions of marketplace components. We have focused on those components that relate in particular to the development of new marketplace request services and inter-enterprise applications. Continuing investigation is aimed at generalisation of the integration server role.

References

1. Abel, D. J., Gaede, V. J., Taylor, K. L., Zhou, X. SMART: Towards Spatial Internet Marketplaces. GeoInformatica 3(2), (1999) pp. 141–164 331
2. Balakrishnan, R. A Service Framework for Dynamic e-Services Interaction, in: *4th Enterprise Distributed Object Computing Conference (EDOC 2000)*, Japan, (2000) pp. 28–37 329
3. Comella-Dorda, S., Wallnau, K., Seacord, R. C., Robert, J., A Survey of Legacy System Modernization Approaches. CMU/SEI-2000-TN-003. (CMU/SEI, 2000). 341
4. Devereux, D., Power, R., The Image Analyst's Workbench, in: *10th Australasian Remote Sensing and Photogrammetry Conference*, Adelaide, 2001, pp. 623-635. 330
5. ESRI. ESRI Shapefile Technical Description. Available at: URL http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf (1998). 338
6. Ferraiolo, J., Scalable Vector Graphics 1.0 Specification. Available at: URL http://www.w3.org/TR/SVG. (2000). 338
7. Jacobsen, H.-A., Gunther, O., and Riessen, G., Component leasing on the World Wide Web, Netnomics, vol. 2, (2000) pp. 191–219 342
8. MapInfo Corporation. MapX 4.0 Object Model. Available at: URL http://www.mapinfo.com/community/free/library/mapx40_objectmodel.pdf (2000). 339
9. Mills, J., Brand, M., and Elmasri, R., AeroWEB: An information infrastructure for the supply chain, in: *Information Infrastructure Systems for Manufacturing II (DIISM'98)*, (2000) pp. 323–336. 342
10. Moore, K. W. and Oliver, L. G., Spot-Lite On-Line, in: *9th Australasian Remote Sensing and Photogrammetry Conference*, Sydney, 1998. 330
11. ODP, Open Distributed Processing Reference Model. ISO/IEC. Available at: URL http://www.iso.ch/RM-ODP 341
12. Papazoglou, M. P., Laufmann, S. C., and Sellis, T. K., An Organizational Framework for Cooperating Intelligent Information Systems. IJICIS 1(1), (1992) pp. 169–202 342
13. Rumbaugh, J., Jacobson, I., and Booch, G., *The Unified Modeling Language Reference Manual*, Reading MA: Addison Wesley, 1999. 330
14. Yang, J., Papazoglou, M. P., Interoperation Support for Electronic Business, Communications of the ACM, 43(6), (2000) pp. 39–47 329

Investigating the Evolution of Electronic Markets

John Debenham and Simeon Simoff

Faculty of Information Technology, University of Technology, Sydney
{debenham,simeon}@it.uts.edu.au

Abstract. Markets evolve through 'entrepreneurial' intervention which is based on intuition and on timely information. An electronic market has been constructed in the laboratory as a collaborative virtual environment to identify timely entrepreneurial information for e-markets. This information is distilled from individual signals in the markets themselves and from signals observed on the Internet. Distributed, concurrent, time-constrained data mining methods are managed using business process management technology to extract timely, reliable information from this inherently unreliable environment.

1 Introduction

E-business is growing fast and with its growth there is a need for a better understanding of the mechanisms that govern e-markets and the way that they evolve. An electronic market has been constructed in the laboratory as a collaborative virtual environment or virtual 'place'. This market is designed to support investigations into how electronic markets will evolve. It supports a comprehensive range of basic market transactions. These market transactions are managed using business process management technology. Market evolution takes place as the result of entrepreneurial intervention which may either be through the creation of new goods, services, needs or requirements, or through the novel combination of existing ones. Entrepreneurial intervention relies on intuition and on information. In this market, a suite of data mining tools are available to the would-be entrepreneur to 'discover' this information. To be of use to an entrepreneur these tools needto produce information and to estimate the value (including the validity) of that information within a set time. This information is distilled from individual signals in the markets themselves and from signals observed on the unreliable, information-overloaded Internet. This electronic market is designed to support experiments in business-to-business (B2B) markets and in business-to-consumer (B2C) markets.

The term „entrepreneur" is used here in the sense of [1]: „The aspect of knowledge which is crucially relevant to entrepreneurship is not so much the substantive knowledge of market data as alertness, the 'knowledge'of where to find market data.". In general usage, the term entrepreneur refers to various aspects of business including the characteristics of successful and innovative business people and firms. The *entrepreneurial opportunity discovery process* lies at the heart of market innovation

© Springer-Verlag Berlin Heidelberg 2001

and evolution, in which new knowledge is created by bringing together partial knowledge of different elements of a market.

The overall conceptual framework for this market place is shown in Fig. 1. It has been constructed as a collaborative virtual environment. At the *e-market place* layer, basic market transactions are managed as industry processes using a multiagent business process management system. These transactions include the creation of particular markets, the buying and selling of goods, and the packaging of goods, services, needs and requirements. The basic market transactions take place between a set of 'major actor classes' in the market—the decision-makers, which are described in Sec. 2. These transactions are monitored by 'supporting actor classes' that include data mining actors—these are described in Sec. 3. At the *market evolution* layer the entrepreneurial transactions are also managed by business process management technology. This includes the timely extraction of reliable information and knowledge—as described in Sec. 4—as well as entrepreneurial intervention. At the *market information* layer, atomic signals from information bots and other sources feed into the data mining actors, the reliability of these atomic signals is evaluated before the signals are combined by the process management system at the market evolution layer. The implementation of the market as a virtual 'place' is described in Sec. 5.

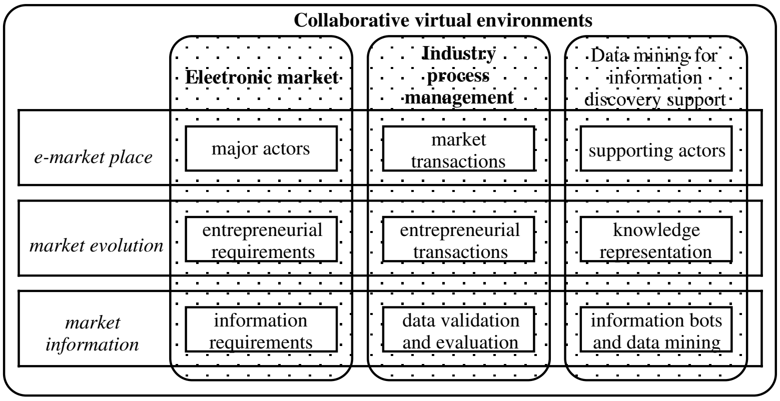

Collaborative virtual environments
Electronic market
Industry process management
Data mining for information discovery support
e-market place
major actors
market transactions
supporting actors
market evolution
entrepreneurial requirements
entrepreneurial transactions
knowledge representation
market information
information requirements
data validation and evaluation
information bots and data mining

Fig. 1. Conceptual framework integrating the domains involved

2 Electronic Markets

E-markets enable the study of the market evolution process using intelligent software and the vast resources of the Internet. First, e-markets are in themselves a fruitful arena for hunting entrepreneurial opportunities—witness the proliferation of new market forms and players [2], [3] and the characterisation of the new network economy as an „opportunity factory" [4]. Second smart computer systems may be applied to leverage the rich flows of information in e-markets leading to the discovery of potentially profitable opportunities and thereby invigorate the market evolution process. „[T]he value of the Internet and IT lies in their capacity to store, analyse and communicate information instantly, anywhere at negligible cost" [Economist, 27 Sep. 2000].

There is also a wealth of established work in economic theory, that describes the behaviour of rational agents performing basic transactions in traditional markets. For example, this work includes the theory of auctions, the theory of bargaining, the theory of contracts and applied negotiation theory [5]. This work may not necessarily describe the behaviour of rational agents operating in electronic markets [6]. Little work has been done on how electronic market places will evolve, although the capacity of the vast amount of information that will reside in those market places, and on the Internet generally, to assist the evolution process is self-evident.

The environment of electronic markets is quite different to the traditional market environment, and so the scope for entrepreneurial activity in electronic markets can not be assumed to be the same. Substantial amounts of data may be acquired quickly and cheaply from the markets themselves. Further, valuable signals may be observed by searching the Internet, by reading news feeds, by watching corporate pages, and by reading background market data such as stock exchange data. If all of these individual signals can be distilled into meaningful information then it may be of use to an entrepreneur. The scope of possible entrepreneurial intervention will determine, and is determined by, the range of the support provided.

The basic market transactions take place in the electronic market place between a set of *major actor classes*; these are based on an extension of the model given in [3]. There are eight major actor classes—see Fig. 2. A central logical component is an „e-exchange“ in which individual markets are created within which deals are done. This is actor class is essentially an „empty virtual box“ for holding markets and for advertising their existence. The *buyer* and *seller* classes include those wishing to buy or sell any form of good or service, or any combination of goods or services, in a market. The remaining actor classes are all entrepreneurial and so are directly instrumental in market evolution. Members of the *content aggregator* actor class act as forward aggregators, they package goods and services, possibly from different sellers, and place these packages in markets in the e-exchange. Members of the *solution providers* class act as intermediaries in the negotiation of contracts and the development of business relationships. *E-speculators* take short term positions in markets in the e-exchange. Sell-side *asset exchanges* exchange or share assets between sellers. *Specialist originators* act as reverse aggregators, they coordinate and package orders for goods and services on behalf of various buyers. In our e-market model, the major actors, shown in Fig.2, are provided with distilled information by the supporting actors, which utilise avariety of data mining and information discovery methods.

3 Data Mining and Knowledge Discovery

E-market is a killer domain for the new generation of data analysis and supporting techniques, collectively labeled as data mining methods. The methods have been successfully applied to stock market analysis, predictions and other financial and market analysis applications [7]. The application of data mining methods in e-business to date has predominantly been done within the framework of the B2C model. This research combines both e-commerce and „classical“ data mining methods [8]. Rather than following the „classical“ sequence, the data mining process in the e-market place is organized as a concurrent process supported by a number of specialised agents. There are seven supporting actor classes, illustrated in Fig. 3. The central actor, the *information kiosk* handles all the internal data and information extracted from this data , notifying the other actors of what is available in the format that they can understand. *Scanners* are the data collectors, responsible for monitoring company web sites, bringing news text and image data, retrieving data from stock markets and other signals. *Preprocessors* are basically data filters and formatters. They convert signal data to particular format and packed it in a „ready-to-use" format, adding information about the format, size, time/date stamp and other service information. *Miners* are the classes that perform the actual data mining exercise. Each miner implements particular machine learning method (the idea is ananalogy of the approach taken by [9] in the development of their class library). Such schema permits to realise combined and/or hierarchical learning schemata, when the output produced by one miner becomes an input to another mining agent. *Composer* plays additional coordinating role between the mining agents. The *info-provider* and *info-visualiser* are supporting actors, which actually provide the distilled information to the major actors and human entrepreneurs.

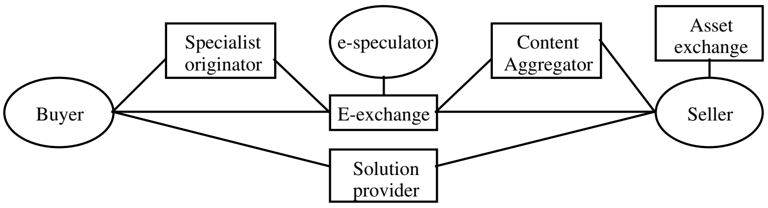


Fig. 2. The major actor classes

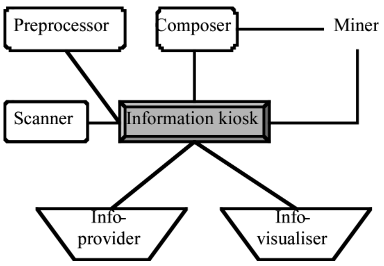


Fig. 3. Supporting actor classes

An example of supporting actors in action is illustrated in Fig. 4. The initial data set, a Web page, supplied by the scanner, is preprocessed by the preprocessor in some structured form. The new data set is clustered by a SOM[1] miner, which produces a table of weights for each node. Such table is stored back in the information kiosk. It can be passed to a major actor in this form or the composer can invoke another mining actor, in this case and Apriori miner (an association rule miner which implements a version of the Apriori algorithm (for more details about the different versions of the Apriori algorithm see [8]). On the request from a major actor, a rule provider (a type of info-provider actor) supplies distilled rules. A map visualiser (a

[1] **S**elf-**O**rganising **M**ap – a type of artificial neural network

type of info-visualiser actor) provides an output from the table of weights in a form comprehensible for humans. In this case, each node is displayed as a square in a colour determined by interpreting the node weights as RGB values;thus nodes with similar weights are displayed in similar colours.

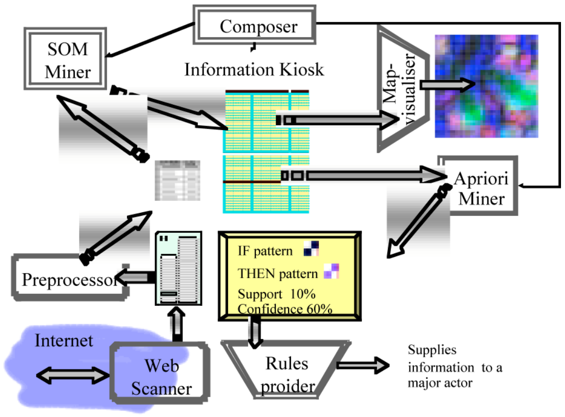


Fig. 4. An example of concurrent agent-based data mining

In the context of providing support for e-market entrepreneurs, such concurrent data mining schema is capable of mining text data from news feeds as well as text and numerical data from financial reports and web pages. The Web scanners in this case are the so-called ‘information bots’, or information agents, that filter news and that watch on-line information for changes in it (for example, NewsHub, NewsTrawler, NewsGuard). The output of the bots, left in the information kiosk, is preprocessed through finer filters and passed as an input to the data mining agents.

The information kiosk supports different knowledge representations. In the example in Fig.4 these are the rule-based format and a neural network weight-matrix. The rules are convenient form for further processing by entrepreneurial support systems thatuse a rule-based inference engine. Such systems can naturally incorporate background knowledge in the form of rules provided by expert entrepreneurs, mixing it with the rules provided by the data mining agents. Regularities, discovered by data mining are communicated and exchanged between the actors in the e-market place, and, if necessary, incorporated in later data mining activities as a priori knowledge. Research results in ontological analysis and engineering [10] [11] is a basis for formal communication languages which support information exchange between the actors in the e-market place.

Thus, the concurrent e-market model integrates two different approaches in data mining – the data driven and the hypothesis-driven approach. In the data-driven approach the major agents are just „absorbing“ the information discovered by the miners. They do not specify what they are looking for before starting to examine the case data. For example, in the text analysis of the news files a text miner can find the frequencies of word occurrences and co-occurrences in the news text. The result of this step is building of an initial representative vocabulary for the news file. In hypothesis-driven approach, major actors specify what they are looking for, for example, they formulate a hypothesis that there is a change in the Board ofDirectors in a large multinational corporation. This means supplying to supporting actors the name of the corporation and at least one person, who is expected to be changed. Such hypotheses can be either refined during the exploration, partially or completely reformulated or finally rejected.

The combination of data-driven and hypothesis driven process aims at providing a mechanism for meeting the tight time constraints. The data driven mining does not require specific request and can be handled in a „prefetched“ mode, when the support agents actually are bringing data and extracting the necessary information, storing the results in the information kiosk. Consequently, some of the information is anticipated to be distilled before it is even required.On the other side, the major actors can access only what is currently distilled in the information kiosk. Thus, to deal with potentially incomplete information, it will be necessary to introduce several measures for an information discovery – importance, relevance to the transaction, potential impact on future transactions.

Managing and synchronizing the activities of the major and supporting actors is not a trivial task. In this e-market place this task is handled by methods and techniques developed inindustry process management.

4 Industry Process Management

The term *industry processes* is used here to refer to processes that are triggered by both e-market transactions and entrepreneurial activities. This is in-line with the term *industry process re-engineering* which concerns the re-engineering of trans-corporate processes as electronically managed processes. Industry process re-engineering addresses the four issues of complexity, interoperability, communication and management. Of these four issues,complexity, communication and management are particularly relevant here. Communication and management are dealt with by an agent architecture based on 'near-failure-proof' plans—as described below.

All e-market transactions are managed as industry processes. The problem of delivering information to an entrepreneur at the right granularity and at the right time is also a process management problem. As is the problem of effecting any intervention that may be triggered indirectly by that information. A single business process system manages both the e-market transactions and the entrepreneurial activities. The complexity in managing e-market transactions stems from the distributed nature of e-markets and from the complex time constraints that govern the operation of e-markets. The complexity in providing entrepreneurial support here stems from the unreliable and unpredictable nature of the Internet, and the problem of delivering something useful and reliable in the required time. This means that a powerful process management system is required, particularly in its capacity to manage heavily constrained and possibly interdependent processes. The system used is a multiagent system that is based on a three-layer, BDI (Belief-Desire-Intention) hybrid agent architecture. A high-level description of this architecture is given below.

One fundamental e-market transaction is the creation by a seller (or buyer) of a particular *market* within the e-exchange. To do this a real (or virtual) seller specifies the goods or services, the market mechanism to be used, and any temporal constraints on the market including when the market will clear. This specification triggers the creation of an instance of the generic *market process* that first requests the creation of certain publicly accessible spaces in the e-exchange in which the market is to be represented. Once this has been achieved this process then advertises the existence of the market and the terms that apply to it, eventually is clears and closes the market. Following the advertisement of the existence of a market, bids (or asks) from a virtual buyer (or seller) trigger an instance of the generic *trading process* which manages the bids, settles the market entry fee (if any), meets the commitments of the bid if it is successful, and manages the retrieval (or delivery) of the goods or services traded. Trading processes may contain time constraints such as„if an attempt to reach an agreement with X on purchasing Y has not succeeded before 2.30pm then abandon that attempt and place a bid for Y in the market Z". These two types of generic process are managed using plans.

The entrepreneurial activities are typically more complex than the e-market transactions. A fundamental entrepreneurial activity is a request for information. Such a request triggers an instance of an *information process* that may invoke any number of a set of basic information discovery and data mining methods (supporting actors) as well as hand-crafted analytical tools. These methods and tools may be applied to either the market data or to the Internet. Information processes may contain critical time constraints such as „get me the best information that you can before 3.00pm". This information is derived from observing a large amount of markettraffic in real-time, and from the inherently-unreliable, information-overloaded Internet. So there are four basic problems here. First, determining where to access the individual signals. Second, assessing some measure of confidence in the validity ofthose signals. Third combining those signals—which may be invalid and which may have been observed at different times and different places—into reliable advice. Fourth to do all this within tight time and cost constraints. Another entrepreneurial activity is *entrepreneurial intervention*. These transactions are innovative in nature. A level of generic support is provided for entrepreneurial activities that fall into the major actor classes of specialist originator, e-speculator, content aggregator and asset exchange. The solution provider class is very broad and hand crafting is usually required. These classes are all illustrated in Fig. 2.

Industry processes are modelled as state and activity charts [12] and are managed by plans that can accommodate failure [13]. The planning system provides the deliberative reasoning mechanism in the agent architecture. The successful execution of a plan in an industry process is not necessarily related to the achievement of that plan's goal. For example, the plan may simply be bad, alternatively the goal may have been achieved by some other plan or externally to the management system. So if a plan executes successfully then its goal may not have been achieved; also, if a plan fails then its goal may have been achieved. This means that each plan should terminate with a *success condition* (SC) that is a procedure whose goal is to determine whether the plan's goal has been achieved. The success condition is the final sub-goal on *every* path through a plan. The success condition is a procedure; the execution of that procedure may succeed (3), fail (7) or abort (**A**). If the execution of the success condition does not succeed then the overall success of the plan is unknown (**?**).

One well reported class of agent architecture is the three-layer, BDI hybrid architecture. One member of this class is the INTERRAP architecture [14], which has its origins in the work of [13]. The process agent architecture described here extends the INTERRAP architecture; it is based on a set of basic concepts or „mental categories". The *conceptual architecture* describes how the agents are specified. The *control architecture* describes how the agents operate.

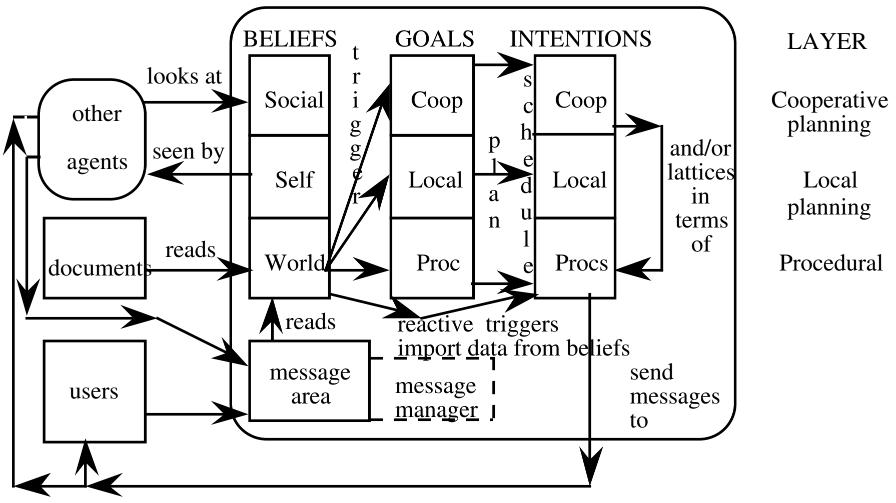


Fig. 5. Conceptual agent architecture

The conceptual architecture is shown in Fig. 5. In that architecture world beliefs are derived *either* from reading messages received from a user, *or* from reading the documents involved in the process instance, *or* from reading messages received from other agents. These activities are fundamentally different. Documents are „passive" in that they are read only when information is required. Users and other agents are „active" in that they send messages when they feel like it. The social beliefs of an agent are the observed self-beliefs of other agents in the system. Beliefs play two roles. First, they can be partly or wholly responsible for the activation of a local or cooperative trigger leading to the agent committing to a goal, and may thus initiate an intention (eg. the execution of a plan to achieve what a message asks, such as „find the value of BHP Corp's ordinary stock"). A cooperative intention will involve other agents such as in a process generated by a content aggregator (eg. will you join with X by providing a service to maintain the computers that X sells thus creating a package deal). This is *deliberative reasoning*. Second, they can be partly or wholly responsible for the activation of a 'procedure trigger 'that will pass data to a partly executed procedure and so may advance the agent's progress in achieving some goal to which it is committed (eg. the message „the value of BHP Corp's ordinary stock at 12.20pm was $12.20" could lead to a belief that activates a procedure trigger). This is *reactive reasoning*. Reactive reasoning also involves „hard-wired" procedures such as a hard-wired procedure trigger that watches for the arrival of messages such as „market Z has closed". The generic agent is implemented in Java; it is implemented as an interpreter of high-level agent specifications. This interpreter enables agents to be built quickly and enables the specification of agent plans to be modified during execution. Implementation as an interpreter also simplifies general maintenance, which only has to deal with high level specifications.

Multiagent technology is an attractive basis for industry process re-engineering [15] [16]. A multiagent system consists of autonomous components that interact with messages. The scalability issue is „solved"—in theory—by establishing a common understanding for inter-agent communication and interaction. KQML (Knowledge

Query and Manipulation Language) is used for inter-agent communication [17]. Specifying an inter-agent communication protocol may be tedious but is not technically complex. Standard XML-based ontologies will enable data to be communicated freely [18] but much work has yet to be done on standards for communicating expertise. Any process management system should take account of the „process knowledge“ and the „performance knowledge. *Process knowledge* is the wisdom that has been accumulated, particularly that which is relevant to the process instance at hand. *Performance knowledge* is knowledge of how effective people, methods and plans are at achieving various things.

5 E-Market Place as a Collaborative Virtual Environment

Market activities are intrinsically social. E-market is a computer-mediated market. There is a number of ways to implement the amalgamation of the actors, representations, analysis methods and techniques presented in the previous sections. We seek a convenient and understandable integration of the underlying technology, which also supports human interactions within the market. Although, by its nature market activities are distributed across borders and places, handling activities in one „logical place“ provides convenient mechanism for observing and collecting data about the behaviours, strategies and activities in the market. A significant effort in the research and development in collaborative virtual environments (CVEs) [19] has gone in the implementation of virtual 'place' models, which offer a convenient way of structuring and handling information, communication, exchange and other functionality in such environments [20]. Virtual worlds are a class of CVE environments that implement one possible elaboration of a virtual place and have the potential to provide scalable professional working environments that integrate communication, interaction, negotiation and other exchange between actors. In our approach the virtual world provides a place where the complete range of market activities is fully supported.

The implementation of an e-market lab for investigating the evolution of an e-market should be able to accommodate new emerging markets, perhaps as new e-market places. Fig. 6 illustrates the idea. The distributed e-market, populated with different e-market places is organised around the concept of an „universe“, populated with numerous „worlds“, respectively. Active Worlds virtual world technology implements this metaphor, providing means not only for developing universes and worlds in 3D representations, but also for populating these spaces with programmable objects.

One aspect of the virtual place model is the establishment of the identity of the agents (whether human or computer) in that place. In the context of our e-market place, the representation of the agent should be understandable to both sides, so to a human in the world an agent may appear as an avatar, as shown in Fig. 7. Thus in oure-market an avatar is a 3D model of an actor and shows where they are, where they are looking, and what gestures they want to communicate.

Technically, the ontology of a virtual world defines the way the agents interact in it. Fig. 8 illustrates the idea. The interaction happens via the Behavior and Transformation Group Nodes, when the representation is placed in a Shape3D node.

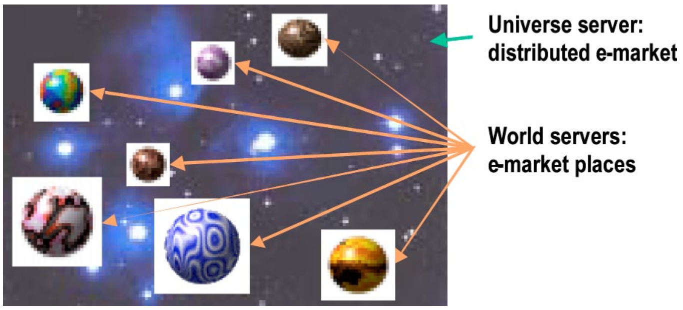


Fig. 6. The idea of Universe/World implementation of a distributed e-market as a collection of e-market places

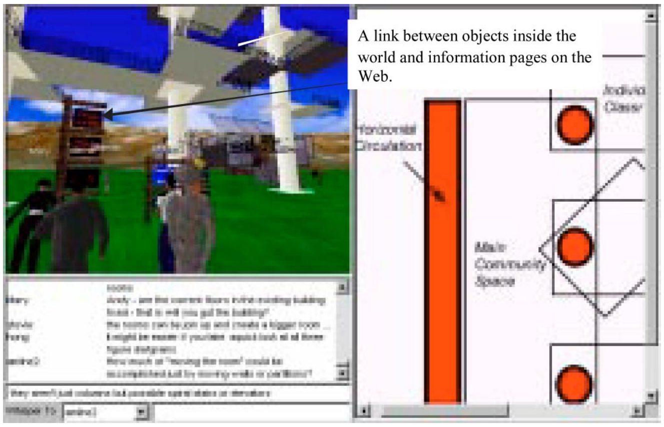

A link between objects inside the world and information pages on the Web.
Horizontal Circulation
Main Community Space
Individual Classr

Fig. 7. An avatar in an e-market place represents an actor

6 Conclusion

One of the innovations in this research is the development of a coherent environment for e-market places and a set of virtual agents for representing the actors. Virtual worlds are a smart way of doing this because they provide a single environment, whose consistent organisation can significantly reduce the cognitive and information overload, and they provide a coherent environment for interactive data mining and decision making. Data (market, communication, and strategic) is warehoused in one place, and is processed by the agents that populate the e-market place. The use of a powerful business process management system to drive all the electronic market transactions unifies the whole market operation. By merging the functionality of e-business and the functionality of the place provided by a virtual world, a new, more effective and coherent environments for e-business will be developed. The development of computational models of market actors, deploying those models in the e-market place, and including them as part of the building blocks for creating e-market places is significant in providing a practical instrument for further research and development in electronic markets.

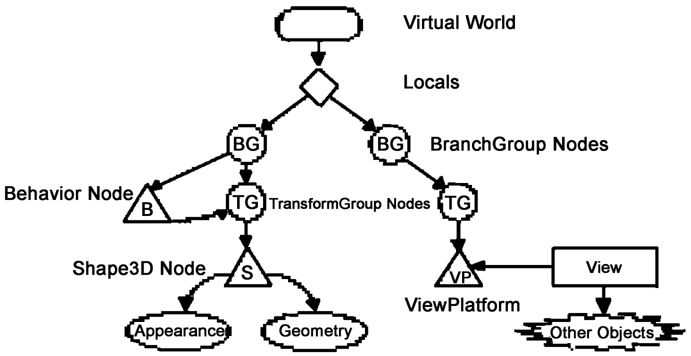


Fig. 8. An ontology of a virtual universe (adapted from Java3D ontology)

References

1. Israel M. Kirzner. Competition & Entrepreneurship. University of Chicago Press, 1973.
2. Kaplan Steven and Mohanbir Sawhney. E-Hubs: The New B2B Marketplace. Harvard Business Review 78 May-June 2000 97-103.
3. R. Wise & D. Morrison. Beyond the Exchange; The Future of B2B. Harvard Business review Nov-Dec 2000, pp86-96.
4. Kelley, Kevin. New Rules for the New Economy. New York, Penguin Books, 2000.
5. Peyman Faratin. Automated Service Negotiation Between Autonomous Computational Agents. PhD dissertation, University of London (Dec 2000).
6. Moshe Tennenholtz. Electronic Commerce: From Economic and Game-Theoretic Models to Working Protocols. Invited paper. Proceedings Sixteenth International Joint Conference on Artificial Intelligence, IJCAI'99, Stockholm, Sweden.
7. B. Kovalerchuk & E. Vityaev. Data Mining in Finance: Advances in Relational and Hybrid Methods. Kluwer, 2000.
8. Han, J. & Kamber, M (2001). Data Mining: Concepts and Techniques, Morgan Kaufmann, San Francisco, CA, 2001.
9. Witten, I. & Frank, E. (2000). Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann Publishers, San Francisco, CA.
10. Uschold, M. and Gruninger, M.: 1996, Ontologies: principles, methods and applications. Knowledge Engineering Review, 11 (2), 1996.
11. Guarino N., Masolo C., and Vetere G., OntoSeek: Content-Based Access to the Web, IEEE Intelligent Systems 14(3), May/June 1999, pp. 70-80
12. Muth, P., Wodtke, D., Weissenfels, J., Kotz D.A. and Weikum, G. „ From Centralized Workflow Specification to Distributed Workflow Execution." In Journal of Intelligent Information Systems (JIIS), Kluwer Academic Publishers, Vol. 10, No. 2, 1998.
13. Rao, A.S. and Georgeff, M.P. „BDI Agents: From Theory to Practice", in proceedings First International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, USA, pp 312—319.
14. MŸller, J.P. „The Design of Intelligent Agents" Springer-Verlag, 1996.
15. Jain, A.K., Aparicio, M. and Singh, M.P. „Agents for Process Coherence in Virtual Enterprises" in Communications of the ACM, Volume 42, No 3, March 1999, pp62-69.
16. Jennings, N.R., Faratin, P., Norman, T.J., O'Brien, P. & Odgers, B. Autonomous Agents for Business Process Management. Int. Journal of Applied Artificial Intelligence 14 (2) 145—189, 2000.

17. Finin, F. Labrou, Y., and Mayfield, J. „KQML as an agent communication language.“ In Jeff Bradshaw (Ed.) Software Agents. MIT Press (1997).
18. Robert Skinstad, R. „Business process integration through XML“. In proceedings XML Europe 2000, Paris, 12-16 June 2000.
19. Capin, T.K., Pandzic, I. S., Magnenat-Thalmann, N. and Thalmann, D. Avatars in Networked Virtual Environments. John Wiley and Sons, Chichester, 1999.
20. J.A. Waterworth. Spaces, places, landscapes and views: experiential design of shared information spaces. In A. J. Munro, K. Hook and D. Benyon (eds), Social Navigation of Information Space, Springer, London, pp. 132-154, 1999.

Coordinating Web-Based Systems with Documents in XMLSpaces

Robert Tolksdorf and Dirk Glaubitz

Technische Universität Berlin, Fachbereich Informatik, FLP/KIT,
Sekr. FR 6–10, Franklinstr. 28/29, D-10587 Berlin, Germany,
{tolk,glaubitz}@cs.tu-berlin.de
http://www.cs.tu-berlin.de/˜tolk

Abstract. We describe an extension to the Linda model of coordination for Web-based applications. It allows XML documents to be stored in a coordination space from where they can be retrieved based on multiple matching relations amongst XML documents, including those given by XML query-languages. XMLSpaces is distributed and supports several distribution policies in an extensible manner. We describe the partial replication schema implemented in detail.

1 Introduction

While the Web has become *the* universal information system worldwide in the first decade of its existence, the progress towards cooperative information systems utilizing the Web for universal access is rather slow. Although there are several technologies like Java or CORBA available, none of these has reached universal acceptance.

A core question in supporting such distributed systems is on the concept applied for the coordination of independent activities in a cooperative whole. This has been the subject of the study of coordination models, languages and systems ([1]). The work described here follows the approach to design a separate *coordination language* ([2]) that deals exclusively with the aspects of the interplay of entities and provides concepts orthogonal to computation.

Most recently, the Web-Standard XML (*Extensible Markup Language*) ([10]) has become *the* format to exchange data markup following application specific syntaxis. It may well be the dominating interchange format for data over networks for the next years. XML data is semi-structured and typed by an external or internal document type or by a minimal grammar inferred from the given document. A DTD (*Document Type Definition*) defines a context-free grammar to which an XML document must adhere. Tags define structures within a document that encapsulate further data. With attributes, certain meta information about the data encapsulated can be expressed. While XML enables collaboration in distributed and open systems by providing common data formats, it is still unclear how components coordinate their work.

The concept of *XMLSpaces* presented in this paper marries the common communication format XML with the coordination language Linda ([3,4]). It

© Springer-Verlag Berlin Heidelberg 2001

aims at providing coordination in Web-based cooperative information systems. XMLSpaces offers a simple yet flexible approach to coordinate components in that context and extends the original Linda-notion with a more flexible matching concept.

This paper is organized as follows. We first look at the coordination language Linda and show how XMLSpaces extends it with XML documents in tuple-fields. We describe the extensible support for multiple matching relations amongst XML documents. After that, we describe the distribution concept applied, with special emphasis on the partial replication scheme implemented, and how distributed events are provided. After brief looks at our implementation platform and related work, we conclude.

2 Linda-Like Coordination

Linda-like languages are based on data-centric coordination models. They introduce the notion of a shared dataspace that decouples partners in communication and collaboration both in space and time ([5]).

The coordination media in Linda is the *tuplespace* which is a multiset of *tuples*. These are in turn ordered lists of unnamed fields typed by a set of primitive types. An example is ⟨10,"Hello"⟩ which consists of an integer and a string.

The tuplespace provides operations that uncouple the coordinated entities in time and space by indirect, anonymous, undirected and asynchronous communication and synchronization. The producer of data can emit a tuple to the tuplespace with the operation *out(⟨10,"Hello"⟩)*. The consumer of that data does not even have to exist at the time it is stored in the space. The producer can terminate before the data is consumed.

To consume some data, a process has to describe what kind of tuple shall be retrieved. This description is called a *template*, which is similar to tuples with the exception, that fields also can contain bottom-elements for each type, eg. ⟨10,?string⟩. These placeholders are called *formals* in contrast to *actuals* which are fields with a value. Given a template, the tuplespace is searched for a *matching* tuple. A *matching relation* on templates and tuples guides that selection.

Retrieving a matching tuple is done by *in(⟨10,?string⟩)* which returns the match and removes it from the space. The primitive *rd(⟨10,?string⟩)* also returns a match but leaves the tuple in the tuplespace. Both primitives *block* until a matching tuple is found, thereby synchronizing the consumer with the production of data.

The Linda matching relation requires the same length of tuples and templates and identical types of the respective fields. For formals in the template, the actual in the tuple has to be of same type, while actuals require the same value in the tuple.

Tuples as in Linda can be considered "primitive data" – there are no higher order values such as nested tuples, no mechanisms to express the intention of typing fields such as names etc. When aiming at coordination in Web-based systems, a richer form of data is needed. It has to be able to capture application specific higher data-structures easily without the need to encode them into primitive fields. The format has to be open so that new types of data can be specified. And it has to be standardized in some way, so that data-items can be exchanged between entities that have different design-origins.

The *Extensible Markup Language* XML ([10]) has recently been defined as a basis for application specific markup for networked documents. It seems to meet all the outlined requirements as a data-representation format to be used in a Linda-like system for open distributed systems. *XMLSpaces* is our system that uses XML documents in addition to ordinary tuple fields to coordinate entities with the Linda-primitives.

3 XMLSpaces

XMLSpaces extends the Linda model in several major aspects:

1. XML documents serve as fields within the coordination space. Thus, ordinary tuples are supported, while XML documents can be represented as one-fielded tuples.
2. A multitude of relations amongst XML documents can be used for matching. While some are supplied, the system is open for extension with further relations.
3. XMLSpaces is distributed so that multiple dataspace servers at different locations form one logic dataspace. A clearly separated distribution policy can easily be tailored to different network restrictions.
4. Distributed events are supported so that clients can be notified when a tuple is added or removed somewhere in the dataspace.

We describe these extensions in this and the following sections.

In XMLSpaces, actual tuple fields can contain an XML document, formal fields can contain some XML document description, such as a query in an XML query language. The matching relation is extended on the field-field level with relations on XML documents and expressions from XML query languages. All Linda operations, and the matching rule for other field-types and tuples are unchanged.

The matching rule to use for XML fields is not statically defined, instead, XMLSpaces supports multiple matching relations on XML documents. The current implementation of XMLSpaces builds on a standard implementation of Linda, namely TSpaces ([11]). It already provides the necessary storage management and the basic implementations for the Linda primitives.

In TSpaces, tuple fields are instances of the class *Field*. It provides a method called *matches(Field f)* that implements the matching-relation amongst fields and returns *true* if it holds. The method is called by the matching method of class *SuperTuple*, which tests for equal length of tuples and templates. Actuals and formals are not modeled as distinguished classes but rather typed according to their use in matching.

XMLSpaces introduces the class *XMLDocField* as a subclass of *Field*. The contents of the field is *typed* as an actual or a formal by fulfilling a Java-interface. If it implements the interface *org.w3c.dom.Document*, it is an actual field containing an XML document. If it implements the interface *XMLMatchable*, it is a formal. Otherwise it is an invalid contents for an *XMLDocField*.

The method *matches* of an *XMLDocField* object tests the polarity of fields for matching. It returns false, when both objects are typed as formals, or when an actual is to be matched against a formal. If both the *XMLDocField*-object and the parameter to *matches* are actuals, a test for equality is performed. Otherwise – if a formal is to be matched against an XML document – the method *xmlMatch* of the formal is used to test a matching relation. Figure 1 shows the resulting class hierarchy.

4 Multiple Matching Relations

The purpose of the interface *XMLMatchable* is to allow for a variety of matching relations amongst XML documents. The template used for *in* and *rd* then, is not relative to the language definition as with Linda, but relative to a relation on XML documents and XML templates that is contained within the template as the implementation of *xmlMatch* in *XMLMatchable*.

The use of multiple matching relations can be an application requirement. We find such a requirement in the Workspaces architecture ([12,13,14]). Workspaces is a Web-based workflow system which combines concepts from the application domain workflow management with standard Internet technology, namely XML and XSL, and with coordination technology.

Steps are the basic kind of activity in Workspaces and represent a unit of work on some application specific XML document. Each step is represented as an XML document that can be distributed individually for interpretation by the XSL-based Workspaces engine. As a result, distributed workflows can be coordinated via the Web. A complete Workflow is described as a graph in an XML-document. It is split into a set of individual steps in an XSL-based compilation step.

Figure 2 shows an example of such a workflow graph – in this case describing the review process for papers submitted to a conference. The application specific documents being manipulated are the papers submitted and reviews forms to be filled out by members of a program committee.

Each step is described in another document, the step document. It contains an XSL script interpreted by the WorkSpaces engine to automatically transform documents, eg. the “Collect reviews” step, to call external applications as with the “Answer questions” step, or to wait for events external to the system as seen for the step “Think and lookup”. XSL scripts are valid XML documents.

The Workspaces engine utilizes XMLSpaces as shown in figure 3. First, some work description is retrieved with an *in* by requesting an XML document that matches the step DTD. Then, the necessary application specific XML document is requested by referring to some identifier in an attribut of a tag of the document.

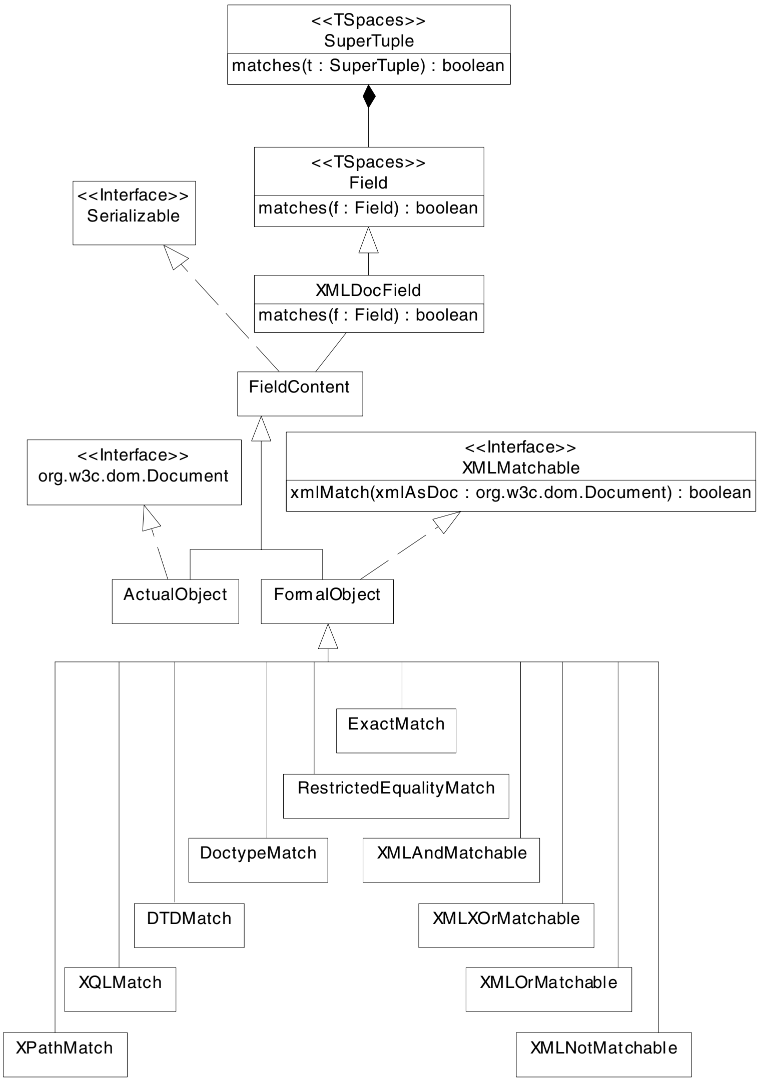
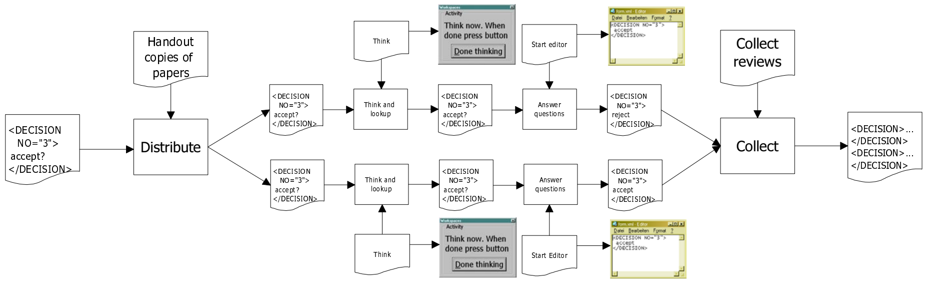


Fig. 1. The class hierarchy for XML documents in tuple fields in UML notation

Then, the actual work is performed as described by the step document. Finally, the changed document is put back to the XMLSpaces with an *out* operation.

During execution of a workflow, one might try to retrieve "*something* to do", which means a document that follows the DTD used for the description of steps.



<DECISION NO="3"> accept? </DECISION>
Handout copies of papers
Distribute
Think
Think and lookup
Think now. When done press button
Done thinking
Start editor
Answer questions
Collect reviews
Collect
<DECISION NO="3"> reject </DECISION>
<DECISION NO="3"> accept </DECISION>
<DECISION>... </DECISION> <DECISION>... </DECISION>
Start Editor

Fig. 2. A WorkSpaces workflow for reviewing conference submissions

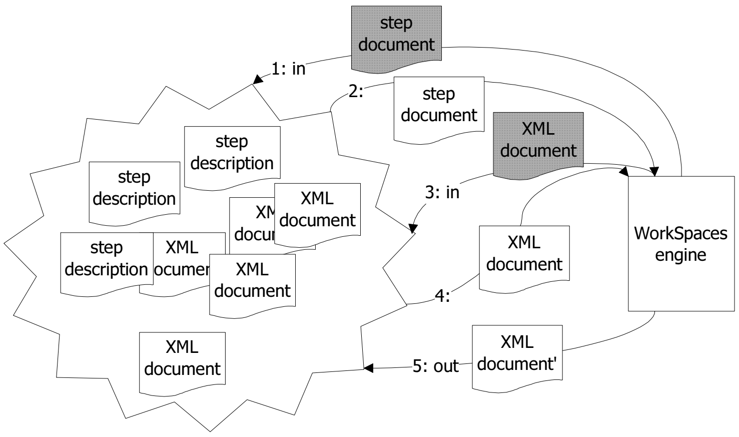


Fig. 3. Usage of XMLSpaces in Workspaces

If, however, a specific task is to be done on a specific application document, one wants that *one* XML document that might be described by an identifier in an attribute. This requirement induces the need for support of multiple relations used in matching.

The individual Workspaces engines benefit from the application of this kind of coordination technology. They are completely uncoupled and can be distributed and mobile. The number of engines participating in the system can be dynamic as new engines can join and leave at whatever time and location they want. The engine that will process future steps does not necessarily have to run when the workflow starts to execute. These attractive advantages of our architecture are due to the use of coordination technology and indicate its usefulness. The decoupled coordination style, indirected by a coordination medium that masks any issues of distribution and synchronization, thus gives huge technical freedom for a distributed and open implementation.

XMLMatchable is also the basis of the extensibility of XMLSpaces with new matching relations. To realize it, a new class has to be provided that implements this interface and tests for the new relation in the *xmlMatch* method. Figure 4 shows an implementation of one matching routine. Here, the XQL-engine from GMD-IPSI is used. The example shows the ease of integration of further matching routines in XMLSpaces.

```
package matchingrelation;
import xmlspaces.XMLMatchable;
import java.io.*;
import org.w3c.dom.Document;
import de.gmd.ipsi.xql.*;
public class XQLMatch implements XMLMatchable{
  String query;
  public XQLMatch(String xqlQuery){
    query = xqlQuery;
  }
  public boolean xmlMatch(Document xmlAsDoc){
    // forward to GMD-IPSI XQL engine
    return XQL.match(query, xmlAsDoc);
  }
}
```

Fig. 4. The implementation of XQL-matching

While the XML standard defines one relation, namely “validates” from an XML document to a DTD, there is a variety of possible other relations amongst XML documents and other forms of templates. These include:

– An XML document can be matched to another one based on equality of contents, or on equality of attributes in elements.
– An XML document can be matched to another one which validates against the same grammar, ie. DTD.
– An XML document can be matched to another one which validates against the same minimal grammar with or without renaming of element- and attribute-names.
– An XML document can be matched to a query expression following the syntax and semantics of those, for example XML-QL, XQL, or XPath/Pointer.

Currently, several relations are implemented in XMLSpaces as shown in table 1. The relations fall into different categories:

– The *equality* relations use several views on what equality of XML documents actually means. For example, whether comments are included in a check or not.

Table 1. Matching relations in XMLSpaces

Relation	Meaning	Tool used
Exact equality	Exact textual equality	DOM interfaces
Restricted equality	Textual equality ignoring comments, processing instructions, etc.	DOM interfaces
DTD	Valid towards a DTD	IBM XML4J Parser
DOCTYPE	Uses specific Doctype name	DOM DocumentType
XPath	Fulfills an XPath expression	Xalan-Java
XQL	Fulfills an XQL expression	GMD-IPSI XQL-Engine
AND	Fulfills two matching relations	–
NOT	Does not fulfill matching relation	–
OR	Fulfills one or two matching relations	–
XOR	Fulfills one matching relation	–

– The *DTD* relations take the relation of a document to a DTD or a doctype name as constituent for matching.
– The *query language* relations build on several existing XML oriented query languages. A query describes a set of XML documents to which the query matches. The query match then is taken as the matching relation in the sense of XMLSpaces. Note that the query languages are of high expressibility, for example, matching for all documents that contain a specific value in some attribute can be formulates as an XPath or XQL expression. While the equality and DTD relations consider a document as a whole, the query relations try to find a match in one part of a document.
– The *connector* relations allow it to build boolean expressions on matching relations whose result gives the final matching relation.

5 Distributed XMLSpaces

In order to make XMLSpaces usable to coordinate wide-area applications, it has to support some form of distribution. It seems to be without question, that centralized coordination platforms suffer from major problems concerning performance and communication bottlenecks, single point of failure ([15]) etc. XMLSpaces supports the integration of XMLSpaces servers at different places into a single logic dataspace.

Distribution of a Linda-like system can be implemented using different distribution schemata which have different efficiency characteristics:

– *Centralized*: One server holds the complete dataspace.
– *Distributed*: All servers hold distinct subsets of the complete dataspace.
– *Full replication*: All servers hold consistent copies of the complete dataspace.
– *Partial replication*: Subsets of servers hold consistent copies of subsets of the dataspace ([6]).

– *Hashing*: Matching tuples and templates are stored at the same server selected by some hashing function ([7]).

XMLSpaces does not prescribe one specific approach but encapsulates the distribution strategy applied in a *distributor object*. It takes care of the registration of a server in the distributed space and offers distributed versions of the coordination primitives at its interface as shown in table 2.

Table 2. The methods of any *Distributor* object

Method:	Explanation
register:	Registers the server in the distributed tuplespace
deregister:	Deregisters the server in the distributed tuplespace
distributedWrite:	Performs an out of the argument tuple to the distributed space
distributedWaitToTake:	Performs an in from the distributed space with the argument template
distributedWaitToRead:	Performs a rd from the distributed space with the argument template
distributedEventRegister:	Registers for the distributed event described in the argument
distributedEventDeRegister:	Deregisters for the distributed event described in the argument

The implementation of the distributor object implements a distribution strategy by the respective versions of these methods. XMLSpaces servers are configured with a distribution strategy at startup. At that time, a respective *Distributor* object is created for the XMLSpaces server, whose *register* method is used to make the server known to remote ones. Exact details of this process are specific to the chosen distribution strategy.

The registration of a server at a remote server is handled by an object of the class *RemoteTSSReception*. Depending on the distribution strategy, it registers the new server and provides it with a reference to an object of class *RemoteTSSReceiver* to which further communication on the distributed coordination primitives is directed.

Figure 5 shows the resulting configuration of objects. After exchanging the initial messages R1 and R2 with the *RemoteTSSReception*, the *Distributor* object now has references to the local space, and to remote spaces as required by the distribution schema. All coordination primitives directed to the local XMLSpaces server are redirected to the distributor object. There, the primitives are implemented by local and remote accesses as given by the strategy.

In the case of a distributed strategy without replication, for example, an *out* is implemented as a simple local *out*, while an in first searches locally and then tries to retrieve a match remotely. For a centralized scheme, the *Distributor*

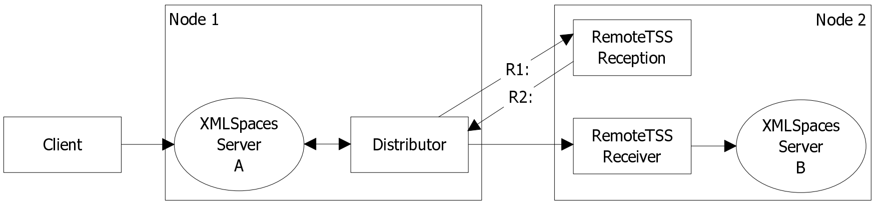


Fig. 5. Server on node 1 connecting to server on node 2

would simply forward the primitives to the central server. Replication schemas are implemented using methods to add, search, lock and delete tuples offered at the *RemoteTSSReceiver* object.

XMLSpaces is *open* in the sense that, with a suited distribution strategy, servers can join and leave at any time. As the distributor object has to know about registered servers, its interface includes respective methods to register and deregister remote servers.

Currently, XMLSpaces includes implementations of the centralized and partial replication strategy. The system can easily be extended by other distributor objects that implement other strategies. The choice of the distribution policy is configured at startup in a configuration file.

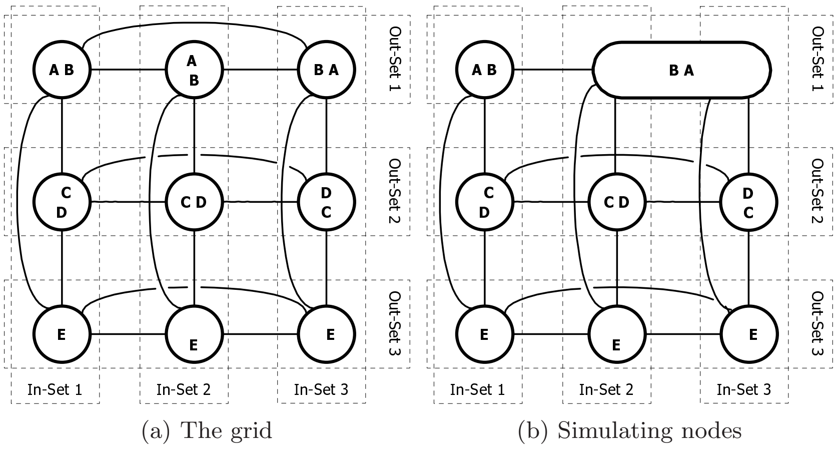


Fig. 6. Partial Replication

The partial replication schema is depicted in figure 6. The nodes in the distributed XMLSpaces each store a subset of the complete contents of the data store. The nodes organized in so called *out-sets* contain identical replicas of a subset as in figure 6(a). An *out*-operation transmits the argument data to all members of the out-set for storage. Every node is at the same time a member of a so called *in-set*. The nodes within one in-set store different subsets of the complete dataspace and the union of their contents represents the whole dataspace. Thus, any *in*- or *rd*-operation asks all nodes in the in-set for a match. In contrast to [8], XMLSpaces does not use a software-bus for communication but members of a set communicate point-to-point.

The structure formed by the sets has to be rectangular – a condition that cannot be upheld in the case of open systems with a varying number of participating nodes. Therefore, the structure has to be retained by *simulating* nodes if necessary. As shown in figure 6(b), one physical node then is part of two out- or in-sets. The reconfiguration of the system in the case of joining and leaving nodes is part of the protocol for joining and leaving nodes.

In XMLSpaces, there is exactly one special node, the *receptionist*, that decides one-at-a-time how new nodes join the structure. It can but does not have to be colocated with a tuplespace server. The receptionist knows about the complete structure of nodes in the system. When a new node joins, it asks the receptionist for its place in the network. Based on heuristics on the fraction of simulated nodes or considerations about the communication efficiency in the in- and out-sets, the receptionist decides on a place in the grid-structure. It informs the new node about the nodes in the target in- and out-sets and perhaps on the address of a simulated node that it shall replace.

There are three possible situations for the placement of a new node:

1. The new node is to replace a simulated node. The new node thus has to copy the state of some existing node in the out-set and stop all operations on the out-set. Then, it registers with all out-set nodes in order to participate in future operations. For registration in the in-set, all nodes there have to be informed that messages shall not be sent to the simulating node but to the new one. When this change is acknowledged, the integration of the new node is complete.
2. The new node is the first one in a new in-set. The receptionist first generates a new in-set which completely consists of simulated nodes, one for each out-set. After that, the new node replaces one of the simulated ones as described.
3. The new node is the first one in a new out-set. It then has to simulate all other nodes in that set. To do so, it registers with all in-sets to complete the integration.

When nodes are to leave the structure, there are two main variants:

1. There are other real nodes in the out-set. In this case, the receptionist decides, what other node will simulate the leaving node and the nodes it simulates in turn. As all nodes are in the same out-set, there is no state to be transferred. The leaving node has to inform the nodes in its in-set about the simulating node. After it deregistered with the nodes in its out-set, it has left the structure.

2. The leaving node is the last real node in the out-set. The leaving node then has to deregister with all in-sets it is member of as a real or simulated node. Any remaining data can be moved to some other out-set and the node has left the structure.

It turns out, that the strategies distributed and full replication as mentioned above are special cases of partial replication: The distributed strategy uses only one in-set while full replication uses only one out-set. They can be implemented by changing the decision of receptionist on the placement of new nodes in that respective one set. XMLSpaces offers respective subclasses of the *Receptionist* class. There is no need to introduce additional subclasses of *Distributor*.

6 Distributed Events

TSpaces supports *events* that can be raised when a tuple is entered or removed from the dataspace. XMLSpaces extends this mechanism to support *distributed events* where clients can register for an event occuring somewhere in the distributed dataspace. As working with distributed events depends on the distribution strategy used, it is implemented in the *Distributor* object.

Supporting distributed events requires mechanisms to register for events, to unregister, notifying about events and handling registered events when integrating new nodes into the grid-structure.

In the case of partial replication, registering and deregistering for events has to be done on all nodes of the in-set. Events are delivered either locally to clients or forwarded to servers in the in-set, that inform their registered clients.

When a new node joins the system by replacing a simulated node, it has to copy all event registrations along with the state. If it forms a new in-set, there are no registrations that have to be considered. Finally, if it forms a new out-set, it has to synchronize with all event registrations on all of its in-sets.

When leaving a system, all locally and remotely registered events have to be deregistered. If the node was the last in its out-set, the registrations can simply be deleted, as no more events will happen. If the leaving node will be simulated afterwards, all registered events have to be transferred to the simulating node.

As with the distribution strategy, it turns out, that distributed and full replication are special cases of partial replication also with respect to events and thus, the distributor implementation can remain unchanged.

7 Implementation

XMLSpaces extends the original Linda conception with XML documents and distribution. It does not change the set of primitives supported nor affect the implemented internal organization of the dataspace. Thus, we have chosen to build on an existing Linda-implementation, namely TSpaces ([11]).

TSpaces is attractive for this purpose, as it is an object-oriented implementation in Java and the XML support can be easily introduced by subclassing.

Also, all issues of server management can be reused for XMLSpaces. In order to support distribution, the original TSpaces implementation had to be extended at some places. TSpaces allowed for a rapid implementation of XMLSpaces focusing on the extensions. However, it could well be exchanged by some other extensible Linda-kernel.

The standard document object model DOM ([16]), level 1, serves as the internal representation of XML documents in actual fields. This leads to a great flexibility to extend XMLSpaces with further matching relations using a standard API. It has shown that the integration of such an engine into XMLSpaces is extremely simple when written in Java and utilizing DOM. If not, some wrapper-object has to be specified in addition. XMLSpaces itself is completely generic towards how the *xmlMatch*-method is implemented and what its semantics are.

As seen in table 1, the huge amount of XML related software provided engines that could directly evaluate the relations on XML documents we are interested in.

8 Related Work

There are some projects documented on extending Linda-like systems with XML documents. However, XMLSpaces seems to be unique in its support for multiple matching relations and its extensibility.

MARS-X ([17]) is an implementation of an extended JavaSpaces ([18]) interface. Here, tuples are represented as Java-objects where instance variables correspond to tuple fields. Such an tuple-object can be externally represented as an element within an XML document. Its representation has to adhere to a tuple-specific DTD. MARS-X closely relates tuples and Java objects and does not look at arbitrary relations amongst XML documents.

XSet ([19]) is an XML database which also incorporates a special matching relation amongst XML documents. Here, queries are XML documents themselves and match any other XML document whose tag structure is a strict superset of that of the query. It should be simple to extend XMLSpaces with this engine.

The note in [20] describes a preversion for an XML-Spaces. However, it provides merely an XML based encoding of tuples and Linda-operations with no significant extension. Apparently, the proposed project was not finished up to now.

TSpaces has some XML support built in ([11]). Here, tuple fields can contain XML documents which are DOM-objects generated from strings. The *scan*-operation provided by TSpaces can take an XQL query and returns all tuples that contain a field with an XML document in which one or more nodes match the XQL query. This ignores the field structure and does not follow the original Linda definition of the matching relation. Also, there is no flexibility to support further relations on XML documents.

9 Conclusion and Outlook

XMLSpaces is a distributed coordination platform that extends the Linda coordination language with the ability to carry XML documents in tuple fields. It is able to support multiple matching relations on XML documents. Both the set of matching relations and the distribution strategy are extensible.

XMLSpaces satisfies the need for better structured coordination data in the Web context by using XML in an open end extensible manner. It has shown that the Linda concept can be extended easily while retaining the original concepts on coordination and a very small core of the coordination language.

Future technological extensions of XMLSpaces include the use of DOM level 2 object model ([21]) to represent XML documents. This standard supports XML Namespaces ([22]) which is necessary to support the full set of XML core specifications in XMLSpaces. Also, this might lead to further matching relations. Issues for extending the functionality are in the areas of security, and fault-tolerance, including extending the transaction concept already existing in TSpaces.

Currently, XMLSpaces is static in its configuration of the distribution policy. A future extension will be support for runtime composition of the system similar to OpenSpaces ([9]). In order to do so, the distributor objects have to be able to establish some "normalized" distribution state from which a new strategy can be built.

Efficiency and scalability of the runtime system has not yet been evaluated. Currently, the system uses RMI for the communication amongst nodes. Using a direct TCP or better UDP protocol for this purpose would speed up the communication.

http://www.cs.tu-berlin.de/˜tolk/xmlspaces gives further details about XMLSpaces.

Acknowledgment

The IBM Almaden Research Center supported the work on XMLSpaces by granting a license to the TSpaces source code.

References

1. G. Papadopoulos and F. Arbab. Coordination models and languages. In Advances in Computers, volume 46: The Engineering of Large Systems. Academic Press, 1998. 356
2. David Gelernter and Nicholas Carriero. Coordination Languages and their Significance. Communications of the ACM, 35(2):97-107, 1992. 356
3. Dirk Glaubitz. Verteilte Linda-artige Koordination mit XML-Dokumenten. Master's thesis, Technische Universität Berlin, 2000. In German. 356
4. Robert Tolksdorf and Dirk Glaubitz. XMLSpaces for Coordination in Web-based Sys-tems. In Proceedings of the Tenth IEEE International Workshops on Enabling Tech-nologies: Infrastructure for Collaborative Enterprises WET ICE 2001. IEEE Computer Society, Press, 2001. 356

5. Nicholas Carriero and David Gelernter. Linda in Context. Communications of the ACM, 32(4):444-458, 1989. 357
6. Craig Fraasen. Intermediate Uniformly Distributed Tuple Space on Transputer Meshes. In J. P. Banâtre and D. Le Mé etayer, editors, Research Directions in High-Level Parallel Programming Languages, number 574 in LNCS, pages 157-173. Springer, 1991. 363
7. Robert Bjornson. Linda on Distributed Memory Multiprocessors. PhD thesis, Yale University Department of Computer Science, 1992. Technical Report 931. 364
8. Robert Tolksdorf. Laura – A Service-Based Coordination Language. Science of Com-puter Programming, Special issue on Coordination Models, Languages, and Applica-tions, 1998. 366
9. Sté ephane Ducasse, Thomas Hofmann, and Oscar Nierstrasz. OpenSpaces: An Object-Oriented Framework For Reconfigurable Coordination Spaces. In Anté onio Porto and Gruia-Catalin Roman, editors, Coordination Languages and Models, LNCS 1906, pages 1-19, Limassol, Cyprus, September 2000. 369
10. World Wide Web Consortium. Extensible markup language (xml) 1.0. W3C Recommendation, 1998. http://www.w3.org/TR/REC-xml. 356, 358
11. P. Wyckoff, S. McLaughry, T. Lehman, and D. Ford. T spaces. *IBM Systems Journal*, 37(3):454–474, 1998. 358, 367, 368
12. Robert Tolksdorf. Coordinating work on the web with workspaces. In *Proceedings of the IEEE Ninth International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises WET ICE 2000*. IEEE Computer Society, Press, 2000. 359
13. Robert Tolksdorf. Coordination technology for workflows on the web: Workspaces. In *Proceedings of the Fourth International Conference on Coordination Models and Languages COORDINATION 2000*, LNCS. Springer-Verlag, 2000. 359
14. Robert Tolksdorf and Marc Stauch. Using xsl to coordinate workflows. In U. Killat and W. Lamersdorf, editors, *Kommunikation in Verteilten Systemen (KiVS)*, Informatik Aktuell, pages 127–138. Springer Verlag, 2001. 359
15. R. Tolksdorf and A. Rowstron. Evaluating fault tolerance methods for large-scale linda-like systems. In *Proceedings of the 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000)*, 2000. 363
16. World Wide Web Consortium. Document object model (dom) level 1 specification. W3C Recommendation, 1998. http://www.w3.org/TR/REC-DOM-Level-1. 368
17. Giacomo Cabri, Letizia Leonardi, and Franco Zambonelli. Xml dataspaces for mobile agent coordination. In *15th ACM Symposium on Applied Computing*, pages 181–188, 2000. 368
18. Eric Freeman, Susanne Hupfer, and Ken Arnold. *JavaSpaces principles, patterns, and practice*. Addison-Wesley, Reading, MA, USA, 1999. 368
19. Ben Yanbin Zhao and Anthony Joseph. The xset xml search engine and xbench xml query benchmark. Technical Report UCB/CSD-00-1112, Computer Science Division (EECS), University of California, Berkeley, 2000. September. 368
20. David Moffat. Xml-tuples and xml-spaces, v0.7. http://uncled.oit.unc.edu/XML/XMLSpaces.html, Mar 1999. 368
21. World Wide Web Consortium. Document object model (dom) level 2 core specification. W3C Recommendation, 2000. http://www.w3.org/TR/DOM-Level-2-Core. 369
22. World Wide Web Consortium. Namespaces in xml. W3C Recommendation, 2000. http://www.w3.org/TR/REC-xml-names. 369

Validating an Access Cost Model for Wide Area Applications[*]

Vladimir Zadorozhny, Louiqa Raschid, Tao Zhan, and Laura Bright

University of Maryland
College Park, MD 20742,
{vladimir,louiqa,taozhan,bright}@umiacs.umd.edu

Abstract. In this paper, we describe a case study in developing an access cost model for WebSources in the context of a wrapper mediator architecture. We document our experiences in validating this model, and note successes and lessons learned. Using experimental data of query feedback from several WebSources, we develop a Catalog and Access Cost model. We identify *WebSource characteristics* of the query feedback that are reflective of the particular WebSource behavior and identify groupings of WebSources based on these characteristics. We also characterize the Access Cost model as having *High* or *Low Prediction Accuracy*, with respect to its ability to predict access costs for the WebSources. We then correlate *WebSource characteristics* and groupings of WebSources with *High* or *Low* prediction accuracy of the model.

1 Introduction

A characteristic of the wide area environment that poses a significant challenge to Wide Area Application (WAA) processing is the dynamic and unpredictable behavior of WANs. Network and server workloads that affect access costs are often transient, and could be affected by parameters such as the Time of Day, the Day of Week etc. Another challenge in WAA processing is the wide variability in the availability of metrics needed for accurate cost estimation. In this paper we focus on the challenges posed by WAA processing with a specific type of source, an autonomous Internet accessible WebSource. These sources typically have *limited query capability*, e.g., they require bindings for input attributes. We consider a wrapper mediator architecture to support WAA processing with WebSources. A wrapper coordinates query processing on each WebSource, and the mediator coordinates queries across multiple WebSources.

There are a number of challenges to be faced in the task of building and validating a catalog of access costs and relevant metrics for WebSources, in the wrapper mediator architecture. We review the relevant research both in the area of database optimization and in network performance.

[*] This research has been partially supported by the Defense Advanced Research Project Agency under grant 01-5-28838, and the National Science Foundation under grant IRI9630102.

© Springer-Verlag Berlin Heidelberg 2001

Autonomous sources typically do not provide any cost information. Research reported in [9,12] assumes that calibration databases can be constructed on remote sources, i.e., the remote sources must accept updates. A generic cost model is calibrated by experiments on a calibrating database created in each source. However, we cannot assume that autonomous WebSources accessible over a public WAN will accept updates, so it may not be feasible to construct a calibration database. The Garlic mediator [20] proposed generic wrapper cost models for wrapped heterogeneous sources. These cost models can be customized by the wrapper implementor. However, they did not consider the WAN environment and WebSource behavior. The dynamic nature of the WAN reflects varying network and server workloads. The Time of day, the Day of week, etc., can significantly affect access costs. Thus, validating these costs has to be an ongoing process. The approach used by the HERMES system [2] is based on query feedback, and can be adapted to model WebSources. However, they do not develop a robust model for learning and prediction that can handle the unpredictable nature of wide area networks. To summarize, none of the previous research on cost estimation has considered transient factors in WAN environments in predicting costs.

We now consider research in the area of network performance. There has been some research on wide area traffic patterns for the Internet [28,26]. SPAND [26] determines network characteristics by sharing measurements that are gathered in a passive manner. While this research is similar, they do not consider transient workload that is affected by The Time of day, the Day of week, etc. They also do not consider workload on the servers where where some query is being executed. Scalable architectures and a variety of caching techniques for improving performance and access [18,23,30] have been proposed. The *Network Weather Service*, NWS [31], is a facility that provides dynamic resource performance forecasts for wide area networks. It uses intrusive resource monitoring; a distributed set of sensors gather data on current network and server conditions. Such data could also be used to predict response time (delay) at WebSources. Finally, IDMaps [11,17] share Internet *distance* between hosts in the form of a distance map; they do not consider the transient workloads that characterize Internet latencies. To summarize, while performance metrics, and techniques to improve performance have been investigated, there has been no research on validating an access cost model for predicting client access on the Internet.

In this paper, we develop an access cost model for WAA processing with WebSources in the wrapper mediator architecture. A special feature of our approach is the use of the Web Prediction Tool (WebPT), a tool that we have developed for online learning and prediction in WANs. The WebPT uses parameters such as the Time of Day and the Day of Week to learn patterns of transient workloads that may affect the access costs for WebSources.

An access cost model for WebSources in a wide area environment must be able to handle the unique challenges posed in this environment. We refer to such a cost model as being *zero assumption* compliant. The *zero assumption* (za) reflects that there are no a priori assumptions about the availability or accuracy of access costs and other metrics. These access costs and metrics must be pro-actively gathered using query feedback and online learning techniques, and must be continually monitored. *za* compliance requires robustness to tolerate noisy environments, i.e., the variability of access costs, and lack of accuracy of prediction of access costs. We have two goals in building our model. One is developing an Access Cost model with the *best possible Prediction Accuracy* for each WebSource. The second goal is determining those *WebSource characteristics* that impact the *Prediction Accuracy*. This goal of correlating *Prediction Accuracy* with readily observable *WebSource characteristics* is key to the scalability of the task of developing *za* compliant optimizers for hundreds of WebSources.

The *WebSource characteristics* that we observe are as follows:

– The significance of the Time and Day on access costs;
– The level of noise or the variability of the access costs;

We develop some hypotheses, based on the *Prediction Accuracy* of the access cost model variants, that correlate *WebSource characteristics* with *Prediction Accuracy*. We validate our hypotheses on a majority of the sources and report on our experiences and successes. We summarize our findings as follows:

– We identify meaningful groupings of *WebSource characteristics*, as well as groupings that do not (typically) characterize the behavior of the observed WebSources.
– We show that the Access Cost model is able to exploit knowledge to improve its prediction accuracy; this is independent of whether the overall accuracy of prediction is high or low for that source.
– We show that we are able to positively correlate groupings of *WebSource characteristics* with the *High* and *Low Prediction Accuracy* of our model.

The contribution of this case study is the ability to use readily observable *WebSource characteristics* to predict the expected accuracy of prediction of an Access Cost model for the WebSource.

This paper is organized as follows: Section 2 describes the Access Cost Model. Section 3 describes classifying WebSources based on their characteristics. Section 4 describes the task of validating the Access Cost model using queries on WebSources. We conclude in Section 5.

2 An Access Cost Model for WebSources

We first describe the metrics that were used to characterize a call to a WebSource. We then describe the WebPT, a tool for predicting response times using online learning. We then describe the Access Cost model.

Recall that a WebSource has limited query capability, i.e. some input attributes must be bound in the query. Thus, a query can be distinguished by the binding pattern and the actual values(s) of the binding(s). The binding pattern for the query, and the value(s) for the binding(s), are important, since they can be used to determine the values of the metrics for a particular query, e.g., the result cardinality. A Catalog stores the values of the metrics described next, for each value(s) of the binding(s) in the query.

2.1 Metrics

We use the following metrics obtained from query feedback to provide estimates of the cost of executing a query in a *WebSource*. We used the Java `URLConnection` class, to send `http` requests to the WebSources, and we used the class interface to gather the following statistics:

- *Remote_Cost* is the time to return the first tuple. The *Remote_Cost* typically reflects the *server* workload and characteristics, e.g., delays at the server due to heavy workload.
- *Download_Cost* is the cost of downloading a page. The *Download_Cost* typically reflects the *network* workload and characteristics.
- *Total_Cost* is the time to return the last tuple. It reflects both network and server workloads.
- *Result_Cardinality* is the number of tuples returned.
- *Page_Size* is the average size of a page containing the result.
- *Num_Accesses* is the number of pages that contain the result.
- *Distribution_of_Values* is the distribution of the domain values for some attribute in the result tuples.

2.2 WebPT

The WebPT or Web (P)rediction (T)ool - is based on an online learning and prediction technique. It collects response times, i.e., *Remote_Cost* or *Download_Cost*, based on query feedback (*qfb*), when a query is submitted to a WebSource. The WebPT organizes this feedback in a nested structure, and uses a simple learning technique to predict the response time for some query. The WebPT is designed for a dynamic environment such as the Web.

Query feedback (qfb) is organized for the WebPT in a nested structure, as shown in Figure 1. The structure is determined by (1) a set of *dimensions* and (2) the *ordering* of dimensions. Typical dimensions are result cardinality, quantity of data, time of day, etc. The quantity of data is obtained using *Num_Accesses* and *Page_Size*. The example WebPT in the figure has three dimensions: Day, Time, and Quantity of data. Each dimension is also associated with a range and scale. For example, the Time dimension may have a range of 24 hours, and a minimum scale of 1 hour. A WebPT will choose some particular *ordering* of these dimensions. In Figure 1, the ordering is Day-Time-Quantity, where Day is most significant. The ordering of the dimensions and the minimum scale for the dimensions will determine the actual WebPT structure, and will impact the learning. The initial structure to organize the query feedback consists of one *cell*. A predicted response time, *PredRT*, is associated with each cell, with initial (default) values.

During the learning process, when a query is submitted to the WebSource, the access costs and metrics of each query feedback *qfb* is identified by a value for Time, Day and Quantity, and actual (measured) response time *QryRT*. As mentioned earlier, this may be *Remote_Cost* or *Download_Cost*. The values for

Time, Day and Quantity of the incoming qfb will be used to identify the matching *cell* of the WebPT structure. The WebPT learning process may then decide, based on the query response time, $QryRT$ of qfb, and the current predicted $PredRT$ of the matching cell, either to split this cell into two or more cells, or to adjust the $PredRT$ using the incoming $QryRT$. This decision is dependent on whether the measured $QryRT$ and the predicted $PredRT$ are within the *allowed deviation* specified for this dimension.

Day	Monday-Friday								Saturday-Sunday
Time	8am-2pm				2pm-8pm			8pm-8 am	12am-12am
Qty	<200K	2-400K	4-600K	>600K	<200K	2-400K	>400K	0-MAX	0-MAX

Monday-Friday			Saturday		Sunday
8am-2pm	2pm-8pm	8pm-8 am	12am-12pm	12pm-12am	12am-12am
			0-MAX	0-MAX	0-MAX

Fig. 1. WebPT Structure for Query Feedback before and after input at 1pm Saturday

Consider the WebPT structure of Figure 1. Suppose there is a qfb on Saturday at 1am, and the $QryRT$ of the qfb, compared to the $PredRT$ of the cell, and the allowed deviation for dimension Day, is such that the cell must be split. Then, the learning algorithm will first split the cell `[(Saturday-Sunday),(12am-12am)]` on the dimension Day, and create two new cells, `[(Saturday),(12am-12am)]` and `[(Sunday),(12am-12am)]`. The incoming qfb is assigned to the first of these cells, and its $QryRT$ will be used to determine $PredRT$, a new prediction for this cell.

Next, depending on the allowed deviation for the Time dimension, the algorithm may split the dimension Time of the first new cell, and create two new cells `[(Saturday),(12am-12pm)]` and `[(Saturday),(12pm-12am)]`. However, the algorithm could make a decision that it will not split this first cell `[(Saturday-Sunday), (12am-12am)]`. Instead, the $QryRT$ of the incoming qfb will be used to adjust $PredRT$ of this cell. Details of the learning algorithm and the features used to tune the learning have been described in detail in [15].

2.3 Access Cost Model

We consider two variants of the Access Cost model. The first variant is a *simple cost model* (SCM), and it exploits all the metrics in the catalog, and the value(s) of query binding(s), to estimate the $Total\_Cost$. However, it does not exploit the WebPT. For each value of query binding, the SCM looks up the $Result\_Cardinality$, the $Remote\_Cost$, the $Download\_Cost$, and the $Num\_Accesses$. It uses a straightforward formula to obtain the $Total\_Cost$ for the query. If there are multiple matching query feedback, corresponding to some value(s) of the binding(s), then averages over all the matching queries are used. For sources where the above statistics vary, depending on the values of particular query bindings, the catalog will store statistics for each binding value. In the absence of statistics for a particular binding value, the cost formula uses the average value of each metric. We note that the SCM is in a similar spirit to [2].

The second variant of the Access Cost model consults the WebPT to obtain *predictions* for the *Remote_Cost* and *Download_Cost*. The wrapper maintains two WebPTs. The WebPT for predicting the *Remote_Cost* uses the Time, Day, and *Result_Cardinality* as dimensions. The WebPT for predicting the *Download_Cost* also uses the quantity of data of one page to be downloaded, determined by the *Page_Size*, as well as Time and Day. Thus, this latter variant, the WebPT cost model, uses the online learning of the WebPT, which uses knowledge of Time and Day to predict the workloads.

3 Experimental Data Collection and WebSource Characteristics

We gathered data from several *WebSources*. The sources are the ACM digital library (ACM) [19]; the CGIAR FishBase (FishBase) [1]; FAA Aviation Safety Data (FAA) [7]; Landings Aviation Search Engines (Aircraft) [10]; the USGS Geographic Names Information System (GNIS) [27]; and the EPA Toxic Releases Inventory Database (EPA) [8]. The sources represent a diversity of queries and answers that we may expect to encounter in a variety of application domains. [1]

To gather data from each source, we submitted queries to the sources at regular intervals. For each query, we recorded several statistics of query feedback, previously described, including *Remote_Cost* (time to first tuple), *Download_Cost*, and *Total_Cost* (time to last tuple). We use the notation `[S, M, L]`, for results with small, medium or large cardinality, for each source. We also varied the test scenario and testing data, using **multiple random** values for the bindings, as well as a **few fixed** values for the bindings, to be discussed later.

We hypothesize that both source and network workloads have a significant impact on latency in a WAN. We further hypothesize that these workloads vary as a function of *Time* and *Day*, and that the WebPT will be able to exploit this knowledge and make more accurate predictions. Thus, our first task is analyzing the query feedback from the WebSources, and classifying them based on their observable characteristics. The *WebSource characteristics* that we considered are as follows:

- The significance of the Time and Day on access costs;
- The level of noise or the variability of the access costs;

We performed standard statistical analysis on the query feedback to test if Time and Day were indeed significant, with respect to access costs. We analyzed *Remote_Cost* (time to first tuple), *Download_Cost*, and *Total_Cost* (time to

[1] We note that none of the sources are corporate sites.

last tuple), as needed. We used two statistical methods, the χ^2 contingency test, and ANOVA [16]. These tests use a variety of analysis to determine whether the independent variable (Time or Day) is significant, and some measure of the level of significance, on the dependent variable (access costs).

We also characterized the level of *noise* or variability of observations, for queries with similar cardinality. As before, we performed the analysis on *Remote_Cost*, *Download_Cost*, and *Total_Cost*. We tested how close the observed query feedback matched with a normal distribution. We used the mean and standard deviation for the observations to compare the level of noise, for the different sources. Figure 2 provides a graphical summary of WebSource *characteristics*. We report on aggregate behavior over the *Remote_Cost*, *Download_Cost*, and *Total_Cost*. In a later section, when we consider cost models where the prediction accuracy is low, we provide a more detailed analysis.

We first consider the significance of Time and Day on the access costs; it is on the left of Figure 2. For several sources, e.g., `ACM`, `EPA`, `FAA` and `FishBase`, both `Time` and `Day` had a (very significant) impact on the query feedback. Thus, we have additional knowledge about these sources that may be exploited by the WebPT. Query feedback from `GNIS` and `Aircraft` showed moderate significance of Time and Day on access costs. For example, for `GNIS`, `Time` was significant but `Day` was not significant. The detailed results of the significance of Time and Day on the access costs are in Appendix A.

Next we consider the level of noise or variability of access costs; it is on the right in Figure 2. All sources exhibited noise, as expected in the WAN environment. The source `EPA` exhibited somewhat less noise, in comparison to `ACM` and `FAA`. The sources `FishBase`, `GNIS` and `Aircraft` exhibit significant noise (variance), with `FishBase` exhibiting the greatest variance.

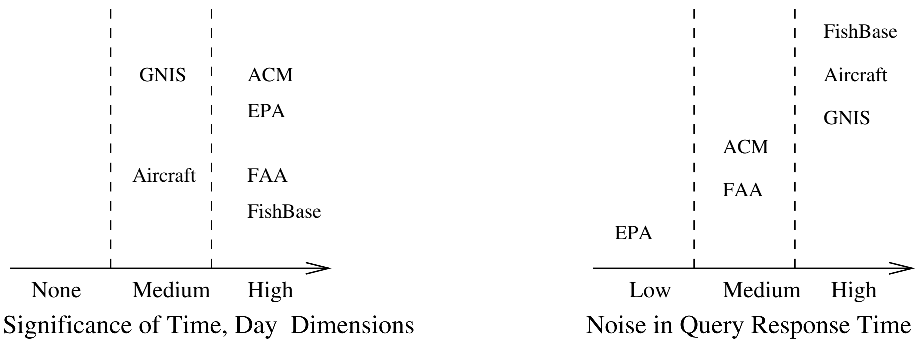


Fig. 2. WebSource Characteristics Using Statistical Analysis of Query Feedback

Our preliminary conclusion is that for a majority of the sources, both `Time` and `Day` appear to have a *very significant* or *significant* impact on query feedback. However, many of the sources display a high level of noise and this may overwhelm the potential to use `Time` and `Day` to predict access costs.

We now consider our first objective *to identify meaningful groupings of WebSource characteristics, as well as groupings that do not (typically) characterize the behavior of the observed WebSources.* Figure 3 shows the groupings.

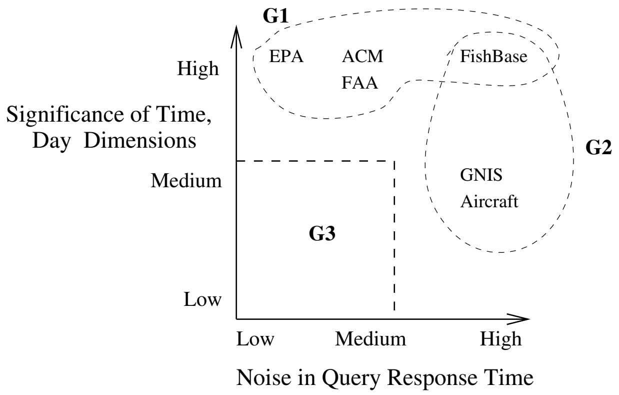

G1
EPA
ACM
FAA
FishBase
High
Significance of Time, Day Dimensions
G2
Medium
GNIS
Aircraft
G3
Low
Low
Medium
High
Noise in Query Response Time

Fig. 3. Grouping WebSource Characteristics

If we consider the upper region of Figure 3, we see that for the four sources for which Time and/or Day are significant, the level of noise or level of variance of access costs can range from low (`EPA`) to high (`FishBase`). Similarly, if we consider the middle and right region of the figure, where the level of noise ranges from medium to high, we see that the level of significance of Time and/or Day on the access costs can also vary from medium (`GNIS, Aircraft`) to high (`ACM, FAA, FishBase`). We note that even when the noise is significant, we do not observe sources where the significance of Time and Day is low. Finally, we observe that there are no sources where the level of noise or variance of the access costs is low, and where the level of significance of Time and/or Day on the access costs *is also low.* We surmise that when the level of noise is low, it cannot overwhelm the level of significance of Time and/or Day on the access costs. We summarize our observed groupings of *WebSource characteristics* as follows:

– Group `G1`: In this group, the significance of Time or Day on the access costs is high, and this is the significant factor; this is independent of the level of noise or variance of access costs.
– Group `G2:`: In this group, the level of noise or variance of access costs is high, and this is the significant factor; this is independent of the significance of Time or Day on the access costs.
– However, when the level of noise is low, we do not observe WebSources where the significance of Time or Day on the access costs is also low. This is region `G3`.

We use these groupings `G1` and `G2` in the next section, where we correlate the *WebSource characteristics* with *za* compliance and the prediction accuracy of the Catalog and Access Cost model.

4 Validating the Access Cost Model

We develop some hypothesis for *za* compliance. This includes the ability to exploit knowledge to improve prediction accuracy, as well as correlating groupings of *WebSource characteristics* with high and low prediction accuracy. These hypothesis are validated using experimental data. We executed external scan queries on several WebSources, repeatedly, over several weeks, and compared the observed and estimated values. The relative error between the observed and estimated values indicates the *accuracy* of prediction. The estimated values were obtained from the two variants of the Access Cost model. We also varied the testing scenario using two variants of the test data. The first scenario involved multiple random bindings for the queries, and the second involved a few fixed bindings for the queries. Experiments were conducted throughout 1999 and 2000.

4.1 Expected Behavior of a *za* Compliant Access Cost Model

Based on our expectations of *za* compliant behavior, we formulate the following hypotheses that must be validated:

- **Hypothesis H1**: A *za* compliant Access Cost model will learn from query feedback and predict response times for queries to WebSources with *High Accuracy*, for the following groupings of *WebSource characteristics*:
  - Parameters such as Time and Day are (statistically) significant and could be used to predict network and server workloads.
  - Less variance of query feedback or low noise so that it does not inhibit the previous factor.
  - This is a subset of the grouping `G1` that does not intersect with `G2`.

  As a corollary, the Access Cost model will have *Low Accuracy* for sources with great variance in response times, i.e., more noise. This is `G2`.

From **H1**, we predict that sources `ACM`, `EPA` and `FAA` are candidates for which a *za* compliant Access Cost model could predict response time with *High Accuracy.* From the corollary, an Access Cost model for `Aircraft`, `GNIS` and `FishBase` would have *Low Accuracy.* This hypothesis addresses the goal of determining groupings of *WebSource characteristics* that impact the *Prediction Accuracy.*

The next two hypotheses address the *za* compliance goal of developing an Access Cost model with the *best possible Prediction Accuracy* for each WebSource, independent of whether the prediction accuracy is high or low.

- **Hypothesis H2:** A *za* compliant Access Cost model will use the WebPT to exploit additional knowledge, e.g., the significance of Time and Day. Thus, the Access Cost model will have higher prediction accuracy for those sources for which these parameters are indeed significant. This benefit would be independent of the level of noise involved. Thus, hypothesis **H2** covers all sources in grouping `G1`, independent of whether the source is in grouping `G2`.

From **H2**, the WebPT model should significantly outperform the SCM model, for sources `ACM`, `EPA`, `FAA` and `FishBase`. There is also a potential that the WebPT could somewhat outperform SCM for `Aircraft` and `GNIS`, since the Time and/or Day is moderately significant for these two sources.

We define a low accuracy test scenario (Lo), where training data and test data correspond to queries with *multiple random* values for input bindings, from all three cardinality groups `[S, M, L]`. We used 50+ values for bindings. In this case, there is a wide variance in the *result cardinality* of the queries, as well as the access costs, across the random set of queries. In contrast, we consider a high accuracy test scenario (Hi), where the training data and test data corresponds to queries with a *few fixed* values for bindings. There is less variance in the access costs across the fixed set of queries. In this case, we used six binding values, two from each cardinality group.

– **Hypothesis H3**: The WebPT(Hi) Access Cost model will exploit the additional knowledge of fixed bindings, and will outperform WebPT(Lo), which is tested with multiple random bindings, with respect to prediction accuracy, independent of the source.

This hypothesis also addresses the goal of developing an Access Cost model with the best possible *Prediction Accuracy* for each WebSource.

4.2 Validating the Ability of the Access Cost Model to Learn

The accuracy of the Access Cost model is defined as the relative error between the *observed* value, and the value *predicted* by that variant of the model. Recall that the WebPT variant exploits Time and Day to predict costs but the simple model SCM does not. Our validation will examine the accuracy of the Catalog and the Access Cost model variant, in predicting *Remote_Cost*, *Download_Cost*, and *Total_Cost*, for the different test scenarios. To validate hypotheses **H1** and **H2**, which test the ability of the model to exploit additional knowledge, we compare the configurations SCM(Hi) and WebPT (Hi), using the *few fixed* values for bindings. A comparison of SCM(Lo) and WebPT (Lo) could also have been be used to validate hypotheses **H1** and **H2**. However, it is possible that the multiple random random bindings of the *low accuracy* test data will have a significant (negative) impact on the predictions of both SCM(Lo) and WebPT (Lo), and so we do not report on these results. To validate hypothesis **H3**, which studies the impact of additional knowledge of fixed bindings, we compare the configurations WebPT(Lo) and WebPT (Hi), both of which exploit the WebPT learning and prediction.

Table 1 summarizes the results of experiments for, for the three variants WebPT(Hi), SCM(Hi), and WebPT(Lo). A value of 1 indicates the method with the *overall greatest accuracy* in predicting the response time for the source, and a value of 3 indicates the method with the *overall least accuracy*, for all the test queries. The notation `all` indicates that the behavior was consistent across all queries of different cardinality. When the behavior was not consistent across all queries, the behavior for each cardinality `[S, M, L]` is provided separately. A value of 1t indicates that two variants tied for *greatest accuracy*.

Table 1 indicates that the `WebPT(Lo)` model was invariably the loser, with a value of "3", and made the worst predictions. While it could exploit knowledge of `Time` and `Day`, since it was trained on queries with multiple random bindings, with a wide variability of access costs, it could not benefit much from query feedback. This validates hypothesis **H3**, and indicates that our Access Cost model variants WebPT(Hi) and SCM(Hi) indeed exploit the added knowledge of fixed bindings.

We now consider hypothesis **H2**, that investigates the ability of the model to exploit the significance of Time and Day on the access costs. From the hypothesis, WebPT(Hi) should significantly outperform SCM(Hi), for sources in grouping `G1`, i.e., `ACM`, `EPA`, `FAA` and `FishBase`, and it should somewhat outperform SCM(Hi) for `Aircraft` and `GNIS`.

For sources, `ACM`, `EPA` and `Aircraft`, we found that WebPT(Hi) made the most accurate predictions, i.e., the corresponding entry had a value of "1", thus, validating **H2**. From Figure 2, `ACM` and `EPA` were in group `G1` where Time and Day were very significant. Time and Day were moderately significant for `Aircraft`. For sources `FAA` and `FishBase` in group `G1`, where the WebPT(Hi) should outperform SCM(Hi), the prediction accuracy of WebPT(Hi) and SCM(Hi) was not consistent across all query bindings. We report on the performance of both methods, for each result cardinality `[S, M, L]`. We note when there is a tie-performance, noted **1t**, of WebPT(Hi) and SCM(Hi). For `FishBase`, SCM(Hi) slightly outperforms WebPT(Hi). For `FAA`, there is a tied performance. We note that `FishBase` is a very noisy source and `FAA` is a fairly noisy source. Thus, while WebPT(Hi) should have exploited Time and Day, and outperformed SCM(Hi), the impact of noise may have hindered the ability of WebPT(Hi) to exploit the significance of Time and Day. This is explored further in a later section.

To summarize, experiments on all sources validate hypothesis **H3**. Sources `ACM` and `EPA`, which are in group `G1` and `Aircraft` validate hypothesis **H2** very strongly. Sources `FAA` and `FishBase` which are also in group `G1` are not inconsistent with **H2** and validate **H2** for some queries. Thus, we conclude that there is a strong correlation between grouping `G1` and hypothesis **H2**, but we did not see perfect alignment.

4.3 Correlation of WebSource Characteristics and Prediction Accuracy

We now consider hypothesis **H1** and the goal of determining those *WebSource characteristics* and *groupings* that impact the *Prediction Accuracy*. We determine the best prediction accuracy for a source, using the best prediction of either WebPT or SCM, and correlate these results with WebSource characteristics, to validate hypothesis **H1**. Table 2 reports on summary statistics of the relative error, for each of the sources. The mean, standard deviation, and the percentage of error within +/- σ (from a normal distribution) are reported, for

Table 1. Ranking of Access Cost Model Variants by Accuracy of Prediction

Source/	ACM	Aircraft	EPA	FAA				FishBase				GNIS			
CostModel	all	all	all	all	S	M	L	all	S	M	L	all	S	M	L
SCM(Hi)	2	2	2	1t	2	2	1	1	1	1	1	1	1	1t	1t
WebPT(Hi)	1	1	1	1t	1	1	2	2	2	2	2	2	2	1t	1t
WebPT(Lo)	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

[1 - method with best prediction accuracy; 3 - method with worst prediction accuracy; 1t - tied performance for best prediction accuracy.]

Table 2. Best and Worst Relative Error of Prediction for WebSources

Source		Method	Relative Error		
			Mean	Std Dev	% in +/- σ
High Accuracy Source	ACM	*WebPT(Hi)	21.89	16.51	70.22
		WebPT(Lo)	63.67	29.64	88.74
	EPA	*WebPT(Hi)	20.09	14.31	67.01
		WebPT(Lo)	44.12	13.50	66.44
	GNIS	*SCM(Hi)	18.84	17.54	81.98
		WebPT(Lo)	48.65	15.56	71
Low Accuracy Source	Aircraft	*WebPT(Hi)	45.18	121.96	95.07
		WebPT(Lo)	449.36	240.26	63
	FAA	*SCM(Hi)	51.66	14.05	69.40
		WebPT(Lo)	55.75	13.17	69.43
	FishBase	*SCM(Hi)	37.76	17.15	70.99
		WebPT(Lo)	78.08	9.53	57.99

[* - method with best prediction accuracy]

the method with the greatest prediction accuracy, noted with a "*". This is either WebPT(Hi) or SCM(Hi). The error for WebPT(Lo) for that source, with the least accuracy, is also reported for comparison purposes.

Recall from hypothesis **H1** that those sources that occur in grouping `G1` but are not in `G2`, e.g., `EPA, ACM,` and `FAA` will have *High Accuracy*. Conversely, sources in grouping `G2`, e.g., `Aircraft, GNIS,` and `FishBase` will have *Low Accuracy*.

From Table 2, `ACM` and `EPA` are indeed *High Accuracy* sources. For these sources, the mean relative error of prediction is around 20%, which is indeed accurate. This result for `ACM` and `EPA` is consistent with **H1** since these two sources are in `G1` but not in `G2`. They are sources with low noise, where Time and Day are significant; they are predicted to be *High Accuracy* sources.

In addition, we observe that `GNIS` which is not in `G1` is *also a High Accuracy* source[2]. This result is extremely encouraging. Time and Day are moderately significant for `GNIS`; this may justify why the source behavior is similar to other sources in grouping `G1`. However, the source is also very noisy, as indicated in Figure 2. Thus, its behavior should be closer to other sources in grouping `G2`, i.e., *Low Accuracy*. The result that this is actually a *High Accuracy* source indicates that a *za* compliant Access Cost model can indeed use the significance of Time and Day to perform well in noisy environments.

From Hypothesis **H1**, noisy sources in grouping `G2` would be *Low Accuracy* sources. From Figure 2, `Aircraft` and `FishBase` are noisy sources iin `G2`. From Table 2, `Aircraft` and `FishBase` are *Low Accuracy* sources. Thus, these sources validate hypothesis **H1**.

Finally, we consider `FAA`. From Figure 2, while this source is a somewhat noisy source, Time and Day are very significant. Thus, this is a source in grouping `G1` which should be *High Accuracy*. We have previously noted that WebPT(Hi) outperforms SCM(Hi) for this source, validating hypothesis **H2** that the Access Cost model can exploit Time and Day to improve prediction accuracy, for sources in grouping `G1`. However, from Table 2, `FAA` is a *Low Accuracy* WebSource. Thus, it appears that the level of noise had a significant detrimental impact on prediction accuracy for this source. The noise appears to have overwhelmed the fact that this source is in grouping `G1` which should be *High Accuracy*.

Thus, the behavior of `FAA` which is in `G1` is unlike `ACM` in `G1`, which overcame moderate noise. It is also unlike `GNIS` which overcame significant noise, and was a *High Accuracy* source, even though it was not in `G1`. We conclude that `FAA` does not strengthen **H1**.

To summarize, with online learning and adequate query feedback, an accurate and *za* compliant Access Cost model and Catalog of access costs can be constructed for WebSources in a WAN environment. All sources in grouping `G1` validated Hypotheses **H2** or were not inconsistent with **H2**. Thus, we showed that the significance of Time and Day on access costs could be exploited to improve the prediction accuracy of the model. All sources also validated Hypotheses **H3**, i.e., the model was able to improve from fixed bindings in the queries.

5 Conclusions and Future Work

In this paper, we report on our experiences in validating an access cost model for Wrappers in the transient WAN environment of WebSources. We formulate several hypotheses for the Access Cost model and we validate these hypotheses using experimental data from several WebSources. We showed that the significance of Time and Day on access costs could be exploited to improve the prediction accuracy of the model. We identified meaningful groupings of *WebSource characteristics*, as well as groupings that do not (typically) characterize the behavior of the observed WebSources. We then correlated combinations of

[2] We note that for some bindings and some queries, the prediction accuracy was sometimes poor.

WebSource characteristics with the *High* and *Low Prediction Accuracy* of our model.

References

1. CGIAR FishBase 99. *http://www.cgiar.org/iclarm/fishbase/search.cfm*. 376
2. S. Adali et al. Query caching and optimization in distributed mediator systems. *Proc. of the ACM Sigmod Conf.*, 1996. 372, 376
3. A. Bairoch and R. Apweiler. The SWISS-PROT protein sequence databank and its supplement TrEMBL. *Nucleic Acids Res*, 1(27):49–54, January 1999. http://www.expasy.ch/sprot.
4. D. Benson, I. Karsch-Mizrachi, D. Lipman, J. Ostell, B. Rapp, and D. Wheeler. GenBank. *Nucleic Acids Res*, 1(28):15–8, January 2000. http://www.ncbi.nlm.nih.gov/Genbank.
5. L. Bright, J-R Gruser, L. Raschid, and M. E. Vidal. A wrapper generation toolkit to specify and construct wrappers for web accessible data sources (websources). *Journal of Computer Systems Science & Engineering. Special Issue: Semantics on the World Wide Web*, 14(2), March 1999.
6. L. Bright, L. Raschid, V. Zadorozhny, and T. Zhan. A comparison of a web prediction tool and a neural network in learning response times for websources using query feedback. *Proceedings of the International Conference on Cooperative Information Systems*, 1999.
7. FAA Aviation Safety Data. *http://nasdac.faa.gov/internet/*. 376
8. EPA Toxic Releases Inventory Database. *http://www.epa.gov/enviro/html/tris/tris_query_java.html*. 376
9. W. Du et al. Query optimization in a heterogeneous dbms. *Proc. of the Very Large Data Bases Conference (VLDB)*, 1992. 372
10. Landings Aviation Search Engines. *http://www.landings.com/_landings/pages/search.html*. 376
11. P. Francis, S. Jamin, V. Paxson, L. Zhang, D. Gryniewicz, and Y . Jin. An architecture for a global internet host distance estimation service. In *Proceedings of IEEE InfoComm*, 1999. 372
12. G. Gardarin et al. *IRO-DB: A Distributed System Federating Object and Relational Databases, In Object-Oriented Multidatabase Systems : A solution for Advanced Applications, Bukhres, O. and Elmagarmid,A.* Prentice Hall, 1996. 372
13. GeneCards. http://bioinformatics.weizmann.ac.il/cards/. Weizmann Institute Genome Center and Bioinformatics Unit.
14. Open System Group. An explanation of the specweb96 benchmark. *http://www.specbench.org/osg/web96/webpaper.html*, 1996.
15. J. R. Gruser, L. Raschid, V. Zadorozhny, and T. Zhan. Learning response time for websources using query feedback and application in query optimization. *To appear in the Very Large Data Base Journal, Special Issue on Databases and the Web. Mendelzon, A. and Atzeni, P., editors.*, 2000. 375
16. SAS Institute Inc. Sas(r) proprietary system for unix(r) environments, release 6.12 (ts060). 377
17. S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavin, and L. Zhang. On the placement of internet instrumentation. In *Proceedings of IEEE InfoComm*, 2000. 372
18. D. Karger, T. Leighton, D. Lewin, and A. Sherman. Web caching with consistent hashing. *Proc. of WWW8*, 1999. 372

19. ACM Digital Library. *http://www.acm.org/dl/Search.html*. 376
20. L. Haas M. Tork Roth, F. Ozcan. Cost models do matter: Providing cost information for diverse data sources in a federated system. *Proc. of the Very Large Data Bases Conference (VLDB)*, 1999. 372
21. L. Ott. *An Introduction to Statistical Methods and Data Analysis*. PWS-Kent, 1984.
22. R. Ramakrishnan P. Seshadri, M. Livny. The case for enhanced abstract data types. *Proc. of VLDB*, 1997.
23. M. Rabinovich and A. Aggarwal. Radar: A scalable architecture for a global web hosting service. *Proc. of WWW8*, 1999. 372
24. M. Rebhan, V. Chalifa-Caspi, J. Prilusky, and D. Lancet. GeneCards: A novel functional genomics compendium with automated data mining and query reformulation support. *Bioinformatics*, July 1998. available at http://bioinformatics.weizmann.ac.il/cards/CABIOS_paper.html.
25. A. Sayal, P. Scheuermann, and P. Vingralek. Selection algorithms for replicated web servers. *Proc. of the Internet Server Performance Workshop (in conjunction with SIGMETRICS'98)*, 1998.
26. M. Stemm, S. Seshan, and R. Katz. A network measurement architecture for adaptive applications. In *Proceedings of IEEE InfoComm*, 2000. 372
27. Geographic Names Information System. *http://mapping.usgs.gov/www/gnis/antform.html*. 376
28. K. Thompson, G. Miller, and R. Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network, November/December*, 1997. 372
29. G. Trent and M. Sake. Webstone: The first generation in http server benchmarking. *http://www.mindcraft.com/webstone/paper.html*, 1995.
30. C. Wills and M. Mikhailov. Towards a better understanding of web resources and server responses for improved caching. *Proc. of WWW8*, 1999. 372
31. R. Wolski. Dynamically forecasting network performance to support dynamic scheduling using the network weather service. *Proc. of the 6th High-Performance Distributed Computing Conference*, 1997. 372

Querying and Splicing of XML Workflows

Vassilis Christophides[1], Richard Hull[2], and Akhil Kumar[2]

[1] Institute of Computer Science, FORTH,
Vassilika Vouton, P.O.Box 1385, GR 711 10, Heraklion, Greece
`christop@ics.forth.gr`

[2] Bell Laboratories, Lucent Technologies
600 Mountain Avenue, Murray Hill, NJ, USA
{`hull,akhil`}`@research.bell-labs.com`

Abstract. In both industry and the research community it is now common to represent workflow schemas and enactments using XML. As a matter of fact, more and more enterprise application integration platforms (e.g., Excelon, Bea, iPlanet, etc.) are using XML to represent workflows within or across enterprise boundaries. In this paper we explore the ability of modern XML query languages (specifically, the W3C XML Algebra underlying the forthcoming XQuery) to query and manipulate workflow schemas and enactments represented as XML data.
The paper focuses on a simple, yet expressive, model called Workflow Query Model (WQM) offering four primary constructs: `sequence`, `choice`, `parallel`, and `loop`. Then three classes of queries are considered against WQM workflows: simple (e.g., to check the status of enactments), traversal (e.g., to check the relationship between tasks, or check the expected running time of a schema), and schema construction (e.g., to create new schemas from a library of workflow components). This querying functionality is quite useful for specifying, enacting and supervising e-services in various e-commerce application contexts and it can be easily specified using the W3C XML Query Algebra.

1 Introduction

During recent years, workflow interoperation has received considerable attention. Numerous research projects and prototypes have been proposed while basic interoperability between various vendor WFMSs has been a subject of standardization efforts by the Object Management Group (see Workflow Management Facility [OMG98]), and the Workflow Management Coalition (see the Xf-XML binding of the WfMC Interface 4 [WMC99]). Recently, XML has become widely accepted as the standard for exchanging not only business data but also information about the enterprise process (e.g., e-services) operating on these data. More and more enterprise application integration platforms (e.g., Excelon, BEA systems, iPlanet, Vitria BusinessWare, icXpertFLOW[1] etc.) and research prototypes [WSFL01,vdAK01,LO01,SGW00,MK00,TAKJ00] are using XML to represent workflow schemas and enactments, within or across enterprise boundaries.

[1] See www.exceloncorp.com, www.bea.com, www.iplanet.com, www.vitria.com and www.icomxpress.com respectively.

© Springer-Verlag Berlin Heidelberg 2001

The use of appropriate XML query languages to access information, about both XML workflow schemas and their enactments, appears as the natural solution for specifying, enacting and supervising e-services within or across organizations. In this paper we show that modern XML query languages (in particular, the W3C XML Algebra [FSW01,FFM$^+$00] underlying the forthcoming XQuery) are expressive enough to issue a variety of useful queries for a simple, but still expressive, workflow model. Furthermore, we propose the incorporation of a number of user-defined functions allowing us to easily navigate through the XML trees representing workflow schemas and enactments.

In this paper we focus on three classes of queries:

Simple: These check on the status of a workflow enactment, i.e., an execution that is in process.

Traversal: These check properties that are more global to workflow schemas and enactments, including issues such as the relationship between tasks (parallel, sequential, exclusive), and the expected (min, max) running time of a schema.

Schema Construction: These queries can be used to construct workflow schemas, using components from a library of "templates" and "base templates"

These queries provide a basis for improving the design and efficiency of workflows, both at compile-time and run-time. For example, potential bottle-necks and race conditions on data usage might be found at compile-time, and potential delays might be detected and averted at run-time.

Given the plethora of models in existing WorkFlow Management Systems (WFMS) we rely in this paper on a pragmatic workflow model allowing us to illustrate concretely how XML query languages can used. This model, called *Workflow Query Model* (*WQM*), involves flowchart constructs with parallelism. More specifically, we focus on four key workflow constructs, namely *sequence*, *choice*, *parallel*, and *loop*. We assume that in a workflow schema these constructs are *properly nested* as per the notion of a structured workflow [KHB00]. Intuitively, this means that there are no go-to's that point into a loop, or that point out of a group of parallel activities. Properly nested workflow schemas can simulate all structured and some unstructured workflows as shown in [KHB00]. As a matter of fact, WQM captures the common features of several commercially available WFMSs, (e.g., Fujitsu i-flow, IBM MQSeries, SAP Business Workflow, FileNet Workflow, Ultimus Workflow[2], or even UML activity diagrams). However, it does not directly address the issues that arise when querying flowchart models based on Petri nets [vdA98] or state charts (e.g., Statemate MAGNUM[3]).

In a nutshell, WQM schemas have a natural tree-based representation, making them ideal for representation and manipulation in XML. With regards to workflow enactments, we follow the same approach for representing them in XML by annotating schema nodes with appropriate status information (e.g., running,

[2] See www.i-flow.com, www.software.ibm.com/ts/mqseries/workflow, www.sap.com, www.filenet.com, www.ultimus1.com, respectively.

[3] See www.ilogix.com.

finished). It should be stressed that workflow querying has received surprisingly little attention so far beyond analysis of workflow logs [KAD98,GT97]. In addition, workflow querying using modern query languages for XML data (e.g., XPath[CD99], Quilt [DC00], XML Query Algebra [FSW01,FFM$^+$00]) is not even addressed in previous research work [WSFL01,LO01,SGW00,MK00,TAKJ00].

This paper is organized as follows. Section 2 provides background and preliminary discussion of our WQM model. Then Section 3 presents how simple status queries can be expressed against enactments created according to the WQM model. Section 4 turns to more complex kinds of traversal queries against both schemas and enactments. Later Section 5 focuses on a novel kind of queries for on demand schema construction. Finally, Section 6 concludes the paper.

2 Background and Preliminaries

This section provides background and preliminary definitions for the rest of the paper. In particular, we present (i) the workflow model we are using to abstract commercial WFMSs, (ii) the variant of XRL [KZ98,vdAK01] we are using to represent in XML workflow schemas and enactments created according to our model, and (iii) how the XML query processor we envision can be integrated into the standard WfMC architecture.

2.1 A Workflow Query Model (WQM)

Figure 1 illustrates a representative workflow schema created using the WQM model which will be used as a running example in the rest of the paper. Task nodes are shown pictorially as rectangles, and the other kinds of nodes are shown using ovals that are labeled with the node type. More precisely, nine kinds of nodes are foreseen to model processes: `start`, `end`, `task`, `split-choice` (with arbitrary out-degree > 1), `join-choice` (with arbitrary in-degree > 1), `split-parallel` (with arbitrary out-degree > 1), `join-parallel` (with arbitrary in-degree > 1), `start-while_do`, and `end_while_do`.

A `split-choice` permits branching in the workflow, where exactly one of the out-edges is taken. The choice may be either deterministic or non-deterministic. In the former case, a specific task out of several alternatives may be chosen after checking a condition. For instance, if two managers can sign an expense approval, the one whose workload is less may be chosen explicitly. On the other hand, a non-deterministic choice may be made as follows. The expense claim could be "offered" to both the managers, and once one of them has accepted the task, it may be withdrawn from the other manager. The parallelism construct is self-explanatory. The `while_do` permits looping over a set of tasks, or in general, over a group of nodes, multiple times. In a WQM workflow schema the `split-choice` and `join-choice` must be matched, as must the `split-parallel` and `join-parallel` nodes, and also `start-while_do` and `end-while_do`. This restriction is primarily to simplify the structure of the workflow schemas that are studied, and thus the queries that need to be expressed.

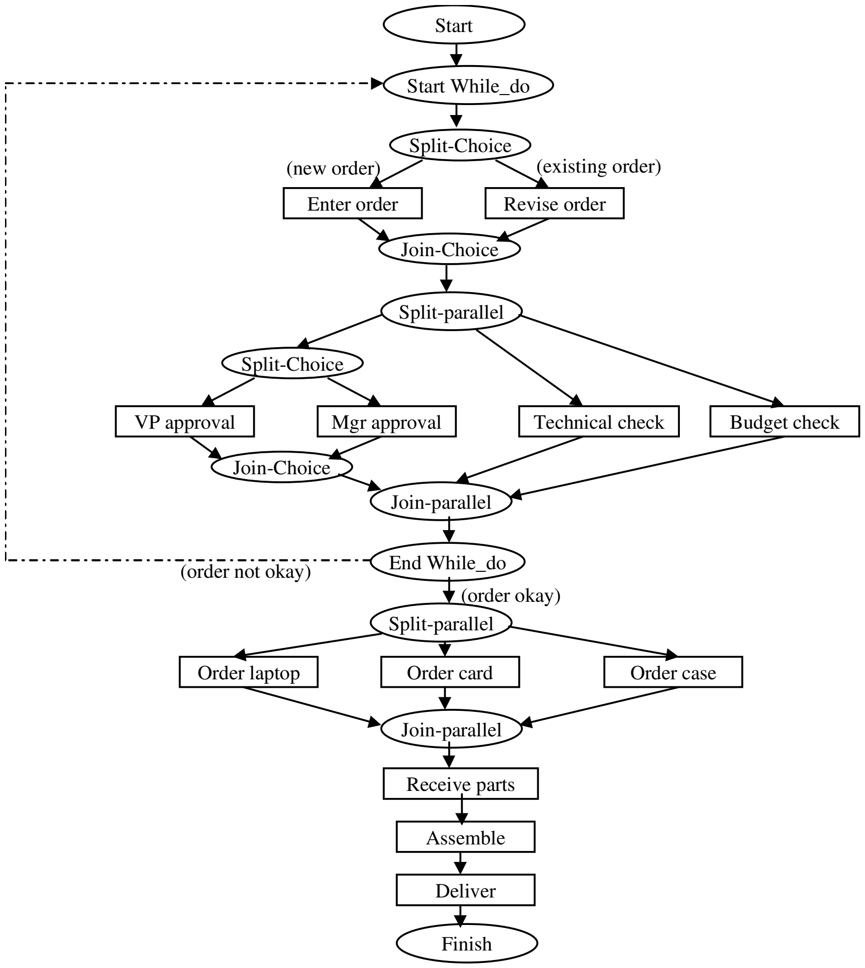


Fig. 1. An example workflow schema using our model

The WQM workflow model also supports specifications indicating which data objects are manipulated by which tasks. In particular, included in a workflow schema is the list of a finite number of named data objects that can be created, read, and/or written by the tasks of the workflow. These data objects can essentially be viewed as global variables that an enactment of the schema can access. For simplicity, we view the data objects as XML documents. Associated with each task is a listing of three disjoint categories of the named data objects, those that the task might create, those it might read, and those it might read and write. Note that exception handling (e.g., out of stock situations, delays, order changes, etc.) is not treated in this paper. For a more theoretical discussion related to structured workflow models, the reader is referred to [KHB00].

2.2 XML Representation of a Schema and an Enactment State

This subsection describes how workflow schemas and enactments can be specified in the WQM model using XML. For this purpose, we are using the type system and syntax of the XML Query Algebra [FSW01,FFM+00] which captures the core semantics of XML Schema [TBMM00,MM00].

First, observe that workflow schemas in the WQM model can be naturally represented as XML trees. Each schema has a root (i.e., a starting element), and

```
let buy_pc1 : WQMEnactment =
  Route [
     Data_list["order_form","engg_spec"],
     Sequence[
       while_do [@count ["0"], @condition ["new-order()? or review(not_okay)?"],
         Sequence [
           Choice[@condition ["new-order()?"]
             Task [ @name ["enter-order"], @status[ "finished" ],
                    @d_create [ "order_form","engg_spec" ]]
             Task [ @name ["revise-order"], @status ["not_ready" ],
                    @d_read [ "engg_spec" ], @d_update [ "order_form"]]
                 ]  (* end Choice *)
             Parallel[
              Choice[@condition ["manager(available)?", @branch ["1"],
                 Task [ @name ["manager-approval"], @status[ "running" ],
                        @d_read [ "engg_spec"],
                        @d_update [ "order_form"]]
                 Task [ @name ["vp-approval"],
                        @d_read [ "engg_spec" ],
                        @d_update [ "order_form"]]
                     ]  (* end Choice *)
               Task  [ @name ["technical-approval"], @status ["ready" ],
                       @d_update [ "order_form","engg_spec"]]
               Task  [ @name ["check-budget"], @status ["finished" ],
                       @d_read["engg_spec"],@d_update [ "order_form"]]
                    ]    (* end Parallel *)
                  ]  (* end Sequence *)
                 ]  (* end while_do *)
           Parallel[
            Task  [ @name ["order-pc"], @status ["not_ready" ],
                    @d_read["engg_spec"],@d_update [ "order_form"]]
            Task  [ @name ["order-card"], @status ["not_ready" ],
                    @d_read["engg_spec"],@d_update [ "order_form"]]
            Task  [ @name ["order-case"], @status ["not_ready" ],
                    @d_read["engg_spec"],@d_update [ "order_form"]]
                 ]    (* end Parallel *)
          Task  [ @name ["receive-parts"], @status ["not_ready" ],
                  @d_update [ "order_form"]]
          Task  [ @name ["assemble-system"], @status ["not_ready" ],
                 @d_read["engg_spec"],@d_update [ "order_form"]]
          Task  [ @name ["deliver"], @status ["not_ready" ],
                 @d_update [ "order_form"]]
              ] (* end  sequence *)
             ] (* end Route *)
```

Fig. 2. The workflow of Figure 1 in syntax of XML Query Algebra

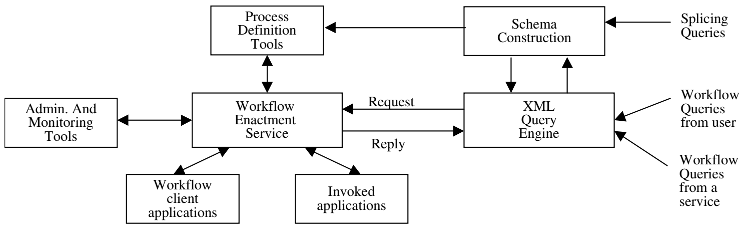

Process Definition Tools
Schema Construction
Splicing Queries
Admin. And Monitoring Tools
Workflow Enactment Service
Request
Reply
XML Query Engine
Workflow Queries from user
Workflow Queries from a service
Workflow client applications
Invoked applications

Fig. 3. Architecture for querying of workflows

may contain several routing elements such as `sequence, parallel, choice, task and slot`, which may be properly nested. Due to space limitations, we are giving in Figure 2 only an enactment example of the workflow schema of Figure 1 defined in WQM.

The WQM XML representation of an enactment essentially captures the information in the coresponding schema with minor variations. Speaking in broad terms, this is achieved by annotating nodes of the tree that encode the workflow schema. In Figure 2 a WQM enactment called `buy_pc1` is presented with a `Route` element which consists of a `Data_list` and `Sequence` subelements. The `Data_list` subelement consists of data object names that are used in this workflow instance. The sequence in turn consists of a `While_do`, a parallel and three task subelements. In general subelements are nested inside their parent elements as comma separated lists enclosed in square brackets. The attributes of an element are also enclosed inside the list associated with it and are preceded by the `@` symbol. The value of an attribute is given in square brackets immediately after its name. Thus, in Figure 2, the `While_do` element has two attributes, `count` and `condition`. The value of the `count` attribute is 0 and the condition attribute has a string value that defines the condition to be checked before executing the loop. Moreover, the `While_do` has a `Sequence` subelement.

The tasks have several attributes, including the `status` of the task, which takes one of the following values: `not_ready`, `ready`, `running`, `finished`. A `not_ready` task is one which is not eligible to start immediately because of precedence constraints. A task is in `ready` state when it becomes eligible to start; however, it has not been assigned to a specific worker. Upon assignment to a worker, it moves into the `running` state and a completed task is in the `finished` state. In this paper, we assume that all tasks initially have the status `not_ready`, although other approaches can be taken. It is conceivable that subtree nodes of a workflow enactment, like `choice` and `parallel` nodes, may also be assigned status values. In our example, we can see that the enactment is currently running the `manager-approval` task. Moreover, the `check-budget` tasks in the first `parallel` node has already finished, and the `technical-approval` task is ready to start. Also, note that the attribute `branch` is associated with a `choice` node and its value gives the `index number` of the branch that was chosen. However, if this attribute is missing for a `choice` node, it means that the enactment has not yet reached this node. Similarly, a `While_do` node has a `count` attribute that stores the number of times the loop has been executed.

2.3 Architecture

This subsection briefly discusses how an XML query engine could fit into an overall workflow management system architecture. To this end, we can see in Figure 3 a slightly modified version of the Workflow Management Coalition (WfMC) architecture. At the heart is an enactment service (also called workflow engine) that runs workflow enactments or instances. Other components that interface with this service are process definition tools, client applications and applications that are invoked by the enactment service. The service also interacts with administration and monitoring tools. In Figure 3 we have singled out the XML query processor as one of these tools. The processor will receive requests from an end user applications regarding a workflow schema or the status of an enactment. It will in turn ask the enactment service to send the corresponding instance(s) to it. Then, the query processor will run the query against the enactment(s) and return an answer to the user. The query processor may also be invoked by another workflow enactment service.

3 Simple Queries

This short section considers straightforward queries that are simple to express *against XML specifications of enactments*. These queries are "local" in two ways: in essence (i) they focus on individual nodes of an XML tree, and (ii) they focus on a snapshot of an enactment. This contrasts with the queries of the following sections, which consider relationships between different parts of an XML tree, and the expected behavior of an enactment over time.

We focus here on some representative queries:

(S.1) what is the status of a task?
(S.2) list all tasks with a given status?
(S.3) what tasks are currently working with a data object?
(S.4) what is the number of times a `While_do` loop has been executed in an enactment?

In addition to giving information about a single enactment, these queries can be used against the set of all currently active enactments, to give a picture of where the "hot spots" are.

All of the queries in this section can be expressed using XPath [CD99]. For example, to answer queries of type S.1, i.e., find the status of task 'shipping', the XPath query would be:

```
/Route//Task[@name="shipping"]/@status
```

The answer to this query would give information about the status of this task as to whether it is `not_ready`, `ready`, `finished`, etc.

A query of type S.3 can also be expressed in a straightforward way in XPath. For instance, to find the tasks that are currently `running` with a data object, say `invoice`, the query would be written as:

```
/Route//Task[@status="running" and (@d_read = "invoice" or
                @d_update="invoice" or @d_create="invoice)]
```

Queries S.2 and S.4 can also be expressed is a similar way using the XPath constructs to get information about the status of the workflow.

4 Traversal Queries

This section considers richer queries, *against both schemas and enactments*, that involve graph traversals. Some of the queries involve traversals of the XML tree representing a schema or enactment, e.g., to determine the relationship between two tasks. Other queries involve traversals of the (annotated) flowchart that embodies the workflow schema (enactment). For both kinds of traversals, we rely heavily on the ability of the XML Query Algebra to express structural recursion, and also use a built-in function for `parent`[4].

The following queries are considered here:

(T.1) what is the relationship of two tasks (parallel, sequential, disjoint)?
(T.2) what is the set of tasks that *may be* touched by a currently active enactment?
(T.3) what is the set of tasks that *must* be touched by a currently active enactment?
(T.4) what is the set of tasks that lie inside a `While_do`?
(T.5) among the tasks inside a `While_do`, what tasks will definitely be executed in each iteration through it?
(T.6) can a given data object be updated further?
(T.7) what is the expected (average, maximum, minimum) estimated running time for a schema/enactment (or a part of it)?

Each of these queries has a variety of uses. Query T.1 can be used to check whether two tasks might compete for updating a data object, or whether two related tasks might occur in `Parallel` (even though that should be prevented). The remaining queries help with analyzing future resource usage of an enactment (or group of enactments), in terms of processing or personnel required, the data objects used, and the timing of when things will complete and when resources will be needed. Queries T.2, through T.6 focus more on *throughput*, while query T.7 focuses on *response-time*. Queries T.2 through T.6 can give information about bottlenecks and “hot spots” where further resources should be allocated over the long term. There is subtle difference between T.2 and T.3 in that T.2 returns a

[4] A built-in function for `ancestor` would provide even more succinctness.

list of all tasks that may be done, while T.3 gives all tasks that will definitely be done. Thus, T.3 returns a subset of T.2, and does not include the tasks inside a Choice element. As a matter of fact, our example queries are approximate in the sense that they do not examine the actual conditions residing at `Choice` or `While_do` nodes. In general, until a branch has been selected at a choice node, it is not possible to predict which path the enactment may take. Thus, all the branches of such a choice element of an enactment are treated as "may be's". Queries T4 and T5 are variants of T2 and T.3, respectively, in that they pertain to tasks that lie inside `While_do` loops. Query T.7 is focused on the short-term and can be used to trigger exceptions if timing thresholds will be exceeded, help re-schedule tasks to alleviate short-term bottlenecks, and to modify future time commitments made about when existing or new enactments might be completed.

As we shall see shortly, to answer these workflow queries we need a structural recursion capability as well as user-defined functions. Both these features are lacking in XPath but they are supported by the XML Query Algebra. Then, the five types of queries can be grouped into three categories and implemented in the XML Algebra.

4.1 Relationship Queries

A query of type T.1 to check the relationship between two tasks can be written in XML Query Algebra, using a user-defined function called *leastCommonAnc*. Depending upon whether the least common ancestor of two tasks is a `Sequence`, `Choice` or `Parallel`, the relationship between these two tasks is accordingly one of sequential, choice or parallel, respectively. We first write an ancestor function `anc` to check if one task is an ancestor of another and then use it in the *leastCommonAnc* function.

```
fun anc(t1:AnyTree;t2:AnyTree):Boolean =
   if (t1 == t2) then true
    else
   let p = parent(t2) do
     if p = () then false
      else anc(t1;p)
```

The above function takes two arguments, t1 and t2, and returns a boolean answer (true, if t1 is an ancestor of t2). Note that the `anc` function uses the parent function proposed in XML Query Algebra. The ancestor function can in turn be used to write a `leastCommonAnc` function which determines the least common ancestor of two tasks, or in general, of any two nodes in the XML tree.

```
fun leastCommonAnc(t1:AnyTree;t2:AnyTree):AnyTree =
   if anc(t1;t2) then
     t1
   else
     ( if anc(t2;t1) then t2
       else leastCommonAnc(parent(t1);parent(t2)))
```

This algorithm also takes two arguments and returns a tree rooted at the least common ancestor node. It works by checking if one of the two nodes is an ancestor of the other; if so, that is the answer. On the other hand, it considers the parents of each of the nodes and checks again for the ancestor relationship between them. Since both the nodes belong to the same tree, a least common ancestor is eventually found. Finally, the next function finds the relationship between two distinct tasks. It calls the `leastCommonAnc` function and prints out the results, i.e., the type of the node which is a least common ancestor. Note that two distinct tasks cannot have a least common ancestor of type `Route` or `While_do`, since those node types have a single child.

```
fun PrintRelation(t1:Task;t2:Task):String{0,*} =
   match  leastCommonAnc(t1;t2)
      case v:Parallel do "Parallel"
      case v:Choice do "Choice"
      case v:Sequence do "Sequence"
      else()
```

4.2 Status Queries

The next three types of queries relate to the status of a running enactment with regards to unfinished tasks and pattern of data object usage. One useful query of type T.2 is to list all the tasks that *may still be* performed on an enactment. This query can be performed by function `maybe-tasks`.

The full text of this function is omitted for lack of space; however, a brief description follows. `maybe-tasks` takes a starting or root node of an enactment as input and walks recursively, depth-first, through the tree, listing the "unfinished" tasks that it encounters. This is accomplished by recursively calling the child nodes of a given node (using the nodes function) until all nodes are exhausted. In the case of a choice node, the branch attribute is checked first. In a running enactment, the value of this attribute is set to the index of the path that is taken after a choice is made. However, an empty value in this variable means that this node has not been reached yet. Hence, all the branches should be listed since any one of them *may be* selected. On the other hand, if a value for the branch attribute is known, then only that branch is pursued. This function may be modified slightly to produce a list of nodes that *must be* traversed to handle a query of type T.3. For this query, the case of a choice node would simply be ignored in the above function. For brevity, we omit the listing of the `mustbe-tasks` function. Moreover, since queries T.4 and T.5 are similar in flavor to T.2 and T.3, respectively, they are left as an exercise for the reader.

Next, we give an example of a query of type T.6 to check if a data object, say d1, will be updated by a subsequent task in the enactment after, say, task t2. The following function `more-updates` takes two input parameters, a tree name corresponding to a running enactment, and a data object name (as a string).

```
fun more-updates(t1:AnyTree; d1:String):Boolean =
  let list1 = maybe-tasks(t1) do
```

```
    not(empty(
         for t in list1 do
            match t
               case t':Task do
                 where contains(t'/@d_update/data(),d1) do t'
               else ()
        ))
```

The above function calls the `maybe-tasks` function to make a list of tasks that may still follow further in a running enactment. Then for each task in the list, it checks if the task writes to the data object of interest. If a match is found then a true value is returned, else the answer is false.

4.3 Time Computation Queries

Lastly, as an example of a type T.7 query, consider the computation of the expected flow time for an enactment. The following `flow_time_left` function, also expressed in the XML Query Algebra, computes the expected time for the completion of an enactment, starting from a given enactment state[5].

```
fun flow_time_left(t1:AnyTree):Float =
  match  t1
   case p:Parallel do max(for c1 in nodes(p) do flow_time_left(c1))
   case c:Choice do dot_product(c/@probs/data(); c)
   case s:Sequence do sum(for c1 in nodes(s) do flow_time_left(c1))
   case t:Task do if t/@status/data() != "finished" do t/@time/data()
                                                           else do 0
   case w:While_do do w/@repeat_factor/data() * flow_time_left(nodes(w))
   else ()
```

We assume that an attribute `time` giving the expected time has been assigned for each task. For a `Parallel` step, function `flow_time_left` includes the maximum expected flow time for its children elements; for a `choice` element, the weighted average is computed using a function `dot_product`. This function (not specified here) takes two lists as argument, computes the product of corresponding pairs, and takes the sum of those products. Here we apply `dot_product` on a list of probabilities for the children of the `choice` (stored in choice node attribute `@prob`) and the `flow_time_left` values for the children of the choice. With a `While_do` node, we compute the `flow_time_left` for its child, and then multiply by a `repeat_factor`, which gives the expected number of times the `While_do` iterates. If none of the tasks have `finished` status, then this function will give the total expected flow time for a schema from start to finish.

[5] Variations of this function for minimum and maximum expected time can easily be constructed.

5 Schema Construction

This section illustrates a novel application of query processing to workflow schemas. In particular, we show how the representation of workflow schemas in XML along with emerging XML query languages can be used to help support the automated construction of workflow schemas. The central idea of the approach is based on a form of hierarchical planning [EHN94], in which workflow schema templates of differing granularity are selected from a repository and then expanded by filling in the slots of those templates appropriately, using additional templates (some of which have no slots), also from the repository. This approach can be used for the on-demand, automated production of specialized workflow schemas, as illustrated in this section. It can also be generalized for use in connection with e-services composition, that is, combining e-services that are provided over the internet and/or the telephony and wireless networks.

To summarize, this section describes how to specify queries of the form:

(C.1) Build a workflow schema from a template and a set of other templates that have been selected for the slots of that template

```
WQMTemplate [
  @name [ "PC_purchase1" ],
  Route [
    Data_list [ "pc_model", "assembler_address", "pc_invoice", ... ],
    Sequence [
      Parallel_sync [
        Slot [ filler_name [ "buy_pc_template" ] ],
        Slot [ filler_name [ "buy_modem_template" ] ],
      ],         (* end Parallel_sync *)
      Slot [ filler_name [ "assembly_template" ] ],
    ]            (* end Sequence *)
  ]              (* end Route *)
]                (* end Template *)
```

(a) PC_purchase1, example template with slots for purchasing and assembling a PC

```
WQMTemplate [
  @name [ "buy_from_Micron1" ],
  @params [ "model", "ship_to", "invoice" ]
  Route [
    Sequence [
      Task [ @name [ "send_order_to_Micron" ],
             @address [ "buy_Micron1.exe" ],
             @d_read [ "model", "ship_to" ],
             @d_update [ "invoice" ]
      ],         (* end Task *)
      ...        (* more Tasks *)
    ]            (* end Sequence *)
  ]              (* end Route *)
]                (* end Template *)
```

(b) buy_Micron1, an example base template for purchasing a PC from Micron

Fig. 4. Example templates from a repository mytemplatedb

The overall approach to building workflow schemas was introduced in [CHKS01]. Here we focus almost exclusively on how the XML Query Algebra is used to support the approach.

5.1 Overview of Approach and Example Application

We view workflow schema construction as a form of *workflow mediation*, because of its analogy with database mediation [Wie92]. However, in our context mediation focuses primarily on enterprise processes, rather than on enterprise data. A workflow mediator can be used to help insulate one organization from the intricacies of interacting with multiple other organizations that together provide some coherent family of e-services. For example, a representative (workflow) mediator might be used by an organization such as Lucent to substantially automate the selection and purchase of PCs (including outsourced assembly and shipping).

Workflow mediators include three main modules, for Planning, Execution, and Data Transformation. The *Planning module* builds workflow schemas based on goals to be achieved (e.g., investigate possible PCs to be purchased; execute the purchase and assembly of selected PCs). The outputs of the Planning process are workflow schemas expressed in XML (and in our current context, in WQM); these are executed by the *Execution module*. The *Data Transformation module* is essentially an XML query processor, which is used by the other two modules.

5.2 Workflow Templates

This and the next subsection together provide an illustration of how the Planning module creates workflow schemas, using the technique of *schema splicing*. This subsection illustrates the pieces of workflow schema that are used, and the next one illustrates how they are spliced together.

In WQM, workflow schemas can be completely specified, or might be *templates* which provide the high-level specification of a workflow but include `Slot` elements where selected other templates can be inserted. Templates without slot elements are called *base templates*. Figure 4(a) shows `PC_purchase1`, an extremely simplified template that might be used for purchasing PCs. Figure 4(b) shows `buy_from_Micron1`, a simplified base template that might be used to fill the `buy_pc_template` slot of `PC_purchase1`. In `PC_purchase1`, a sequence of two activities will occur. The first activity involves the parallel execution of two inserted tasks (which will involve ordering items from a PC vendor and a modem vendor, respectively, and having them shipped to an appropriate location). The second activity consists in executing an inserted task, which asks an assembler to assemble the PC and modem, and ship to another location.

The base template `buy_from_Micron1` may be used to purchase a PC from vendor Micron. The schema for this template includes a task which is to send the `pc_model` and `ship_to` address to Micron, and receive an `invoice` in return. The attribute `params` permit parameter passing in and out of the template; in general these refer to the XML data that a workflow enactment will manipulate. The attribute `address` of the task gives the executable file containing the program required to perform the task. This program might invoke a wrapper or other functionalities that are resident in the mediator, or provided by external systems. In practice, this schema should include actions to be taken if Micron doesn't respond in a timely fashion, or if the PC model is not available, and actions to ensure that the receiving party eventually receives the PC in good condition.

5.3 Schema Splicing

In general, "schema splicing" refers to the creation of new workflow schemas from existing workflow schemas and templates. Figure 5(a) illustrates a mapping (with type `Mapping`, not specified here) that provides the correspondence between the parameters appearing in `PC_purchase1` and either scalar values (e.g., `"Millennia MAX XP"`) or selected elements (e.g., `buy_from_ Micron1`) of a template repository. We assume that the Planning module has selected `PC_purchase1` and constructed `mymapping` assigning the parameters of `PC_purchase1`.

Figure 5(b) shows an XML Algebra function that can be used to replace slots of a template with other templates, choosing the other templates according to a mapping such as `mymapping`. Figure 5(c) shows a query that will fill in the template `PC_purchase1` using `mymapping`. We now describe the operation of this function and query in some detail[6].

The function `fill_slots` of Figure 5(b) takes in a `WQMtemplate` and "instructions" for how to fill the `Slots`, and produces a WQM workflow schema that has no `Slots`, i.e., a fully grounded WQM workflow schema. As with some of our previous queries, `fill_slots` involves structural recursion at the outer layer. Speaking intuitively, the function `fill_slots` performs a depth-first traversal of input `x` (of type `WQMTemplate`), and produces a new, somewhat modified "copy" of `x`. For each element of type `Slot` (i.e., in the first case of the `case` expression) it chooses an appropriate base template from `templatedb` and places it in the output XML object. In the opposite, (i.e., in the `else` part of the `case` expression), the function copies the high-level structure into the output XML object, and recursively calls `fill_slots` on the inner parts of that element.

We now give more detail on function `fill_slots`. Given a `Slot` element `x'`, we first find the element `m` in `mapping` that matches with `x'` (i.e., such that `m/filler_name = x'/filler_name`). For that element `m` we next find the matching template `t` in `templatedb` (i.e., such that `t/@name = m/value`). Finally, we generate from template `t` the content needed for the output XML object. This content is basically the routing part of `t`, namely `t/route/*`. However, we apply a substitution function `fill_params` (not defined here), that replaces all occurrences of the parameters of `t` listed in the `@params` attribute by the parameter values indicated in the `fill_params` element of map `m`. For example, this would replace parameters `"model"`, `"ship_to"`, and `"invoice"` in `buy_from_Micron1`

[6] We assume for now that each slot of the input template will be filled with base templates. That is, we do not have to recursively fill in the slots of templates used to fill other slots.

```
let mymapping : Mapping =
  map [ filler_name [ "pc_model" ],
        value_name [ "Millennia MAX XP" ] ],
  map [ filler_name [ "buy_pc_template" ],
        value_name [ "buy_from_Micron1" ],
        param_assign [ "pc_model", "assembler_address", "pc_invoice" ] ],
  ...
```

(a) Part of example mapping used for schema splicing

```
fun fill_slots(x:WQMTemplate;
               mapping:Mapping;
               templatedb:TemplateDB ) : WQMTemplate =
  match x
    case x':Slot do
      for m in mapping do
        where m/filler_name = x'/filler_name do
          for t in templatedb do
            where t/@name = m/value_name do
              fill_params(t/route/*; m/param_assign)
    else~(name(x))[for v in nodes(x) do
                      fill_slots(v; mapping; templatedb)]
```

(b) Recursive function used to splice templates together

```
query
  for x in mytemplatedb do
    where x/@name/data() = "PC_purchase1" do
      fill_slots(x/*; mymapping; mytemplatedb)
```

(c) Query used to construct a particular schema

Fig. 5. Data, function, and query for constructing a workflow schema

by the parameter values "pc_model", "assembler_address", and "pc_invoice" in the first slot of PC_purchase1.

The else clause of fill_slots involves some aspects of XML Algebra that we haven't seen before. Intuitively, as function fill_slots performs the traversal of input x, when it comes to a non-Slot node n then it produces a node in the output XML object that has the same name as n (this is achieved by the construct ~(name(x))). The contents of the replacement of node n are specified within the square brackets. To produce this content, fill_slots is called on each of the children of the original n.

The query of Figure 5 invokes fill_slots on the template from mytemplatedb whose name is PC_purchase1, i.e., the template of Figure 4(a). The output of this query will be a (ground) workflow schema for buying a PC from Micron. The mapping mymapping should be used when enacting this schema, so that the input data objects for the enactment (e.g., pc_model) will be initialized appropriately.

In the above example just a single layer of hierarchical planning was used, i.e., slots in the top-level template (PC_purchase1) were filled with base templates. In general, multiple layers can be used, such that slots of the top-level template are in turn filled with successive lower-level templates, and finally with base templates at the lowest level of the hierarchy. In terms of the XML Query Algebra function `fill_slots`, this recursion is achieved by having a call to `fill_slots` embedded within the function `fill_params`.

6 Conclusions

We believe that using XML to represent and query workflow schemas and enactments opens new perspectives in workflow management within or across organizations. In particular, it permits improving the design and efficiency of workflows, both at compile-time and run-time. For example, potential bottle-necks and race conditions on data usage might be found at compile-time using analysis queries, and potential delays might be detected and averted at run-time by using status queries. This functionality is required for specifying, enacting and supervising e-services in various e-commerce application contexts.

To this end, we have introduced a simple yet expressive workflow model, called WQM, that uses flowchart-based constructs that are "properly nested". Properly nested schemas are particularly convenient for querying with XML query languages. Indeed, we have demonstrated the power of the XML Query Algebra (offering structural recursion and user-defined functions for traversing workflow schemas) to express a broad range of queries against properly nested workflow schemas and enactments, and to dynamically construct workflow schemas. It should be finally stressed that WQM abstracts several commercially available WFMSs, based on procedural or associative task-based models. There is ongoing work on WQM in order to enable a generic wrapping in XML of heterogeneous workflow models. It remains open how our techniques can be applied in the context of flowchart-based workflow models that are not properly nested, or those based on Petri nets or state charts.

Acknowledgements

The authors thank Jérôme Siméon and Geliang Tong for help in specifying many of the queries in this paper in the latest version of the W3C XML Query Algebra. The third author is currently on leave from the University of Colorado, Boulder.

References

CD99. J. Clark and S. DeRose. XML Path Language (XPath). Technical report, World Wide Web Consortium, 1999. W3C Recommendation 16 November 1999. 388, 392

CHKS01. V. Christophides, R. Hull, A. Kumar, and J. Siméon. Workflow mediation using VorteXML. *IEEE Data Engineering Bulletin*, 24(1), March 2001. 398

DC00. D. Florescu D. Chamberlin, J. Robie. Quilt: An xml query language for heterogeneous data sources. In *WebDB'2000*, pages 53–62, Dallas, US., May 2000. 388

EHN94. K. Erol, J. Hendler, and D. S. Nau. Semantics for hierarchical task-network planning. Technical Report CS-TR-3239, Computer Science Department, University of Maryland, 1994. 397

FFM$^+$00. P. Fankhauser, M. Fernandez, A. Malhotra, M. Rys, J. Siméon, and P. Wadler. The XML query algebra. W3C Working Draft 07 June 2001. Available at `http://www.w3.org/TR/query-algebra/`. 387, 388, 390

FSW01. M. Fernandez, J. Siméon, and P. Wadler. A semi-monad for semi-structured data. In *Proc. of Intl. Conf. on Database Theory*, 2001. 387, 388, 390

GT97. A. Geppert and D. Tombros. Logging and post-mortem analysis of workflow executions based on event histories. In *Proc. 3rd Intl. Workshop on Rules in Database Systems*, Skoevde, Sweden, June 1997. 388

KHB00. B. Kiepuszewski, A. ter Hofstede and C. Bussler On Structured Workflow Modelling In *Proc. CAISE '00*, Stockholm, Sweden, 2000. 387, 390

KAD98. P. Koksal, S. Arpinar, and A. Dogac. Workflow history management. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 27(1), 1998. 388

KZ98. A. Kumar and L. Zhao. XRL: An extensible routing language for electronic applications. In *Intl. Conf. on Telecommunications and Electronic Commerce*, 1998. 388

LO01. K. Lenz and A. Oberweis. Modeling Interorganizational Workflows with XML Nets In *Proc. of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34)*, January 2001. Available at `http://dlib.computer.org/conferen/hicss/0981/pdf/09817052.pdf`. 386, 388

MM00. M. Maloney and A. Malhotra. XML schema part 2: Datatypes. W3C Recommendation, October 2000. Available at `http://www.w3.org/TR/-xmlschema-2/`. 390

MK00. M. zur Mühlen and F. Klein. AFRICA: Workflow interoperability based on XML-messages In *Proc. of CAiSE*00 Workshop on Infrastructures for Dynamic Business-to-Business Service Outsourcing (IDSO'00)*, Stockholm, June 2000. 386, 388

OMG98. Object Management Group. Workflow management facility, joint submission bom/98-06-07, revised, July 1998. Available at `ftp://ftp.omg.org/-pub/docs/bom/98-06-07.pdf`. 386

SGW00. G. Shegalov, M. Gillmann, and G. Weikum. Xml-enabled workflow management for e-services across heterogeneous platforms. In *1st Workshop on Technologies for E-Services (TES)*, Cairo, Egypt, September 2000. 386, 388

TBMM00. H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML schema part 1: Structures. W3C Recommendation, October 2000. Available at `http://www.w3.org/TR/xmlschema-1/`. 390

TAKJ00. A. Tripathi and T. Ahmed and V. Kakani and S. Jaman., Implementing Distributed Workflow Systems from XML Specifications, Available at `http://www.cs.umn.edu/Ajanta/papers/asa-ma.ps`. 386, 388

vdA98. W. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998. 387

vdAK01. W. van der Aalst and A. Kumar. XML based schema definition for support of inter-organizational workflow. Technical Report in review, CU Boulder, 2001. 386, 388

Wie92. Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, March 1992. 398

WMC99. Workflow Management Coalition. Workflow standard - interoperability Wf-XML binding document number wfmc-tc-1023, April 1999. 386

WSFL01. Web Services Flow Language (WSFL), IBM Corporation. Available at `http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`. 386, 388

On Demand Business-to-Business Integration

Liangzhao Zeng, Boualem Benatallah, and Anne H. H. Ngu

School of Computer Science & Engineering, University of New South Wales
Sydney, NSW 2052, Australia
{zlzhao, boualem}@cse.unsw.edu.au
angu@research.telcordia.com

Abstract. We develop an agent-based cross-enterprise Workflow Management System (WFMS) which can integrate business processes on user's demand. In our approach, business processes are wrapped by service agents. Based on users' requirements, the integration agent contacts the discovery agent to locate appropriate service agents, then negotiates with the service agents about task executions. A cost model is proposed which allows the integration agent to update execution plan and integrate service agents dynamically.

1 Introduction

Leading organizations are embracing business-to-business integration (B2Bi) today, realizing the enormous competitive advantages that this model provides through faster time to market, reduced cycle times and increased customer service. An example of a B2Bi is a financial management system that uses payroll, tax preparation, and cash management as components. The component processes might all be outsourced from business partners.

The fast and dynamic integration of business processes is an essential requirement for organizations to adapt their business practices to the dynamic nature of the Web. B2Bi requires the ability to efficiently discover and exploit business services in a dynamic and constantly growing environment. It also requires the capacity to dynamically establish relationships among business processes.

With respect to meeting the requirements of B2Bi, we identified two relevant emerging technologies: *workflow* and *agent* technologies. Workflow Management Systems (WFMSs) have achieved considerable improvements in critical, contemporary measures of performance, such as cost, quality of service, and speed by coordinating and streamlining complex business processes within large organizations (e.g., health-care, education, banking and finance, manufacturing, and communication). Since the success of using workflow technology in automating internal business processes, there is a trend to apply workflow technology to automate external business processes between the enterprises and their trading partners. Agents have emerged recently as an important paradigm for organizing many classes of distributed applications [5]. Agents are software entities with a high degree of autonomy, pro-activity and adaptivity. These properties make

© Springer-Verlag Berlin Heidelberg 2001

agent technology one of the most important candidates for providing interoperability and interactions in volatile, dynamic and cooperative environments.

Our approach, in the AgFlow project [11], is mainly motivated by the fact that for dynamic and constantly changing business processes, there is a need to integrate business processes on user's demand. The philosophy behind our agent-based approach is that B2Bi craves the workflow technology, but current workflow technology is unable to cope with pro-activeness, dynamic interactions (negotiation), and component autonomy which is what agent-based systems can provide.

The remainder of this paper is organized as follows. In Section 2, we give a brief overview of the AgFlow system. In Section 3, we discuss how to select a workflow execution plan. In Section 4, we discuss dynamic B2Bi. In Section 5, we present our current implementation. Finally, we discuss related work in Section 6.

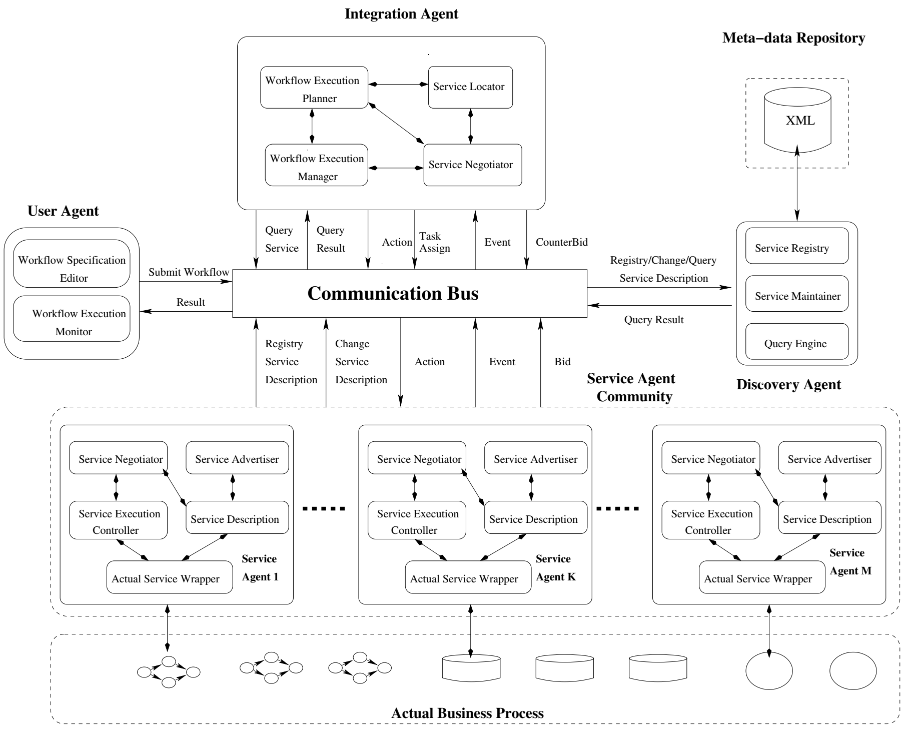


Fig. 1. AgFlow System Architecture

2 The AgFlow: An Overview

The detailed architecture diagram of the AgFlow is presented in Figure 1. There are four types of agents in the system, namely *user agent*, *service agent*, *discovery agent*, and *integration agent*. In the AgFlow, B2Bi is regarded as cross-enterprise workflows which involve multiple enterprises' business processes. For each workflow instance, these four kinds of agents dynamically construct an agent community to execute the workflow.

The user agent provides an interface which allows users to access the AgFlow. There are two types of users in AgFlow: process composers and end users. Process composers can define workflow schemas. End users can create, control and monitor workflow instances. In AgFlow, a workflow schema is defined using a UML statechart diagram. A workflow is textually expressed as an XML document. A service agent encapsulates a proprietary/native service (e.g. flight booking, Java program).

The discovery agent allows each service agent to register its location, properties and service description in a meta-data repository. The meta-data repository contains meta-data that describe, among others, the meaning, type, content, capability and location of the service agents. By searching the repository, the discovery agent can answer query about which service agents can execute a particular task. The integration agent allows the composition of service agents in order to execute the workflow instance. It contacts the discovery agent to locate appropriate service agents. It negotiates with service agents about task executions. This paper focuses on the integration agent. Details about the service discovery agent can be found in [10].

3 Workflow Execution Planning

In this section, we discuss how the integration agent coordinates with other agents to select a workflow execution plan. Here, we assume that the business processes have been wrapped by service agents and the service agents have registered with the discovery agent. In Section 3.1, we discuss the discovery of service agents. In Section 3.2, we present the negotiation between the integration agent and service agents. Details on generating and selecting workflow execution plans are given in Section 3.3 and Section 3.4, respectively.

3.1 Locating Service Agents

When a workflow instance is created, the integration agent parses the workflow description into a set of tasks W[1]:

$$W = \{t_1, t_2, ..., t_n\} \tag{1}$$

[1] Here, we abstract a workflow as a set of tasks. The control flow will be considered during execution of the workflow.

For each task t_i, the integration agent sends a query to the discovery agent to search all the service agents that can execute the task t_i. The search result is a set of service agents A_i. After finishing the queries for each task t_i in W, the integration agent gets a set of 2-tuples C:

$$C = \{< t_1, A_1 >, < t_2, A_2 >, ..., < t_n, A_n >\} \tag{2}$$

$$A = \bigcup_{i=1}^{n} A_i = \{a_1, a_2, ..., a_m\} \tag{3}$$

In (3), the set A represents all service agents that can execute certain task in the workflow W. For each service agent a_i in A, after searching the tuples in C and putting all the tasks that the service agent a_i can execute into a set, we can have C':

$$C' = \{< a_1, T_1 >, < a_2, T_2 >, ..., < a_m, T_m >\} \tag{4}$$

where T_i denotes a set of tasks, the 2-tuple $< a_i, T_i >$ denotes that the service agent a_i can execute all the tasks in T_i. After grouping service agents by tasks that they can execute, we can have C^*:

$$C^* = \{< A'_1, T_1 >, < A'_2, T_2 >, ..., < A'_l, T_l >\} \tag{5}$$

$$\mathbb{T} = \{T_1, T_2, ..., T_l\} \tag{6}$$

In (5), the 2-tuple $< A'_i, T_i >$ denotes that $\forall a \in A'_i$, the service agent a can execute all the tasks in T_i. In (6), $\forall T_i \in \mathbb{T}$, there is at least one service agent that can execute T_i.

3.2 Negotiation between the Integration Agent and Service Agents

For each 2-tuple $< A'_i, T_i >$ in C^*, the integration agent starts a negotiation session N_i to negotiate with every service agent in A'_i about executing the tasks T_i. A negotiation session starts with the integration agent sending a call-for-bid to each service agent in A'_i. The call-for-bid includes specification of the tasks T_i and deadline for responding to the call-for-bid. Once a service agent inspects the call-for-bid from the integration agent, it will decide whether or not to respond with a bid, depending on the task specification, its resources and workload. If a service agent chooses to respond to the call-for-bid, it will send a bid in an XML document. If the bid is for one task (i.e.,$|T_i| = 1$), the service agent will indicate the cost it charges for executing the task and the duration of task execution. If the bid is for a set of tasks (i.e.,$|T_i| > 1$), then the cost and duration will be specified for each task in T_i. Here, we assume that the bid cannot be partially accepted. The integration agent either accepts the whole bid or rejects it.

3.3 Generating Execution Plans

Once bids are available, the integration agent can select and compose the service agents that can be used to execute the workflow. Before giving details about selecting service agents, we introduce the following definitions:

Definition 1 (*Execution Schema*).
$s = \{T_1^*, T_2^*, ..., T_m^*\}$ is an **execution schema** of the workflow $W = \{t_1, t_2, ..., t_n\}$ if:

- $T_i^* \subseteq W$,
- $T_i^* \bigcap T_j^* = \emptyset$, when $i \neq j$,
- $\bigcup_{i=1}^{m} T_i^* = W$, and
- $T_i^* \in \mathbb{T}$.

□

In this definition, all the tasks T_i^* in s are mutually disjoint subsets. The combination of all the tasks T_i^* is the workflow W, and each T_i^* can be executed by different service agents, respectively. So, all the tasks will be executed only once. In fact, the execution schema indicates how to decompose workflow into a set of tasks. For example, $W = \{t_1, t_2, t_3, t_4, t_5\}$, $s = \{T_1^*, T_2^*\}$ is an execution plan of W, if $T_1^* = \{t_1, t_3\}$, $T_2^* = \{t_2, t_4, t_5\}$ and there are at least two service agents that can execute T_1^* and T_2^* respectively.
Definition 2 (*Execution Plan*).
$p = \{< T_1^*, b_1, a_1 >, < T_2^*, b_2, a_2 >, ..., < T_m^*, b_m, a_m >\}$ is an **execution plan** of the execution schema s if:

- For each 3-tuple $< T_i^*, b_i, a_i >$ in p, the service agent a_i sent the bid b_i for the tasks T_i^*.

□

In fact, an execution plan indicates which service agents can be selected to execute the workflow. The integration agent uses the Set Packing algorithm[2][8] to generate a set of execution schemas S from W, C^* and $\mathbb{T}$ (see the definitions in Section 3.1):

$$S = \{s_1, s_2, ..., s_n\} \tag{7}$$

Based on the available bids, for each s_i in S, the integration agent will generate a set of execution plans P_i:

$$P_i = \{p_1, p_2, ..., p_k\} \tag{8}$$

$$P = \bigcup_{i=1}^{n} P_i \tag{9}$$

where P is the set of all the execution plans that can be used to compose service agents.

[2] The Set Packing algorithm is a graph algorithm which is used in applications such as network planning, scheduling, etc. The description of this algorithm is outside the scope of this paper.

3.4 Selecting Execution Plans

In this section, we first discuss the criteria which will be considered when selecting execution plans. Then details about the selection process are given.

Evaluation Criteria. The following criteria will be used to select execution plans:

- **Cost and Time.** The total cost and the total execution time are two basic criteria that we use to select a desired execution plan. For an execution plan p_i, the total execution cost $c(p_i)$ and total execution time $t(p_i)$ are calculated as follows:-

$$c(p_i) = \sum_{k=1}^{m} c\_bid_k \tag{10}$$

$$t(p_i) = CPT(t\_bid_1, t\_bid_2, ..., t\_bid_m) \tag{11}$$

  In (10), total execution cost is the sum of the execution cost (i.e., c_bid_i) in every bid from the execution plan p_i. In (11), t_bid_k denotes the work duration in every bid from the execution plan p_i. The function CPT uses the Critical Path algorithm[3][9] to calculate the execution plan's total execution time. Ideally, the workflow is expected to be executed in minimal time with minimal cost. Unfortunately, it is not always possible to have such an execution plan. We will explain how to handle this situation in Section 3.4.2.

- **Size of agent community.** For different execution plans, the corresponding agent communities could have different size Z_p (i.e., the number of agents in community), since some service agents can execute one task, while other service agents can execute a set of tasks. Bigger size agent community will have more communication overhead.

- **Reliability of service agent.** Service agents' reliability is an important issue to be considered when selecting execution plans. Basically, the reliability has twofold meanings: whether the service agent can finish tasks and whether it can finish tasks on time as specified in its bid. Here, R represents the service agent's reliability:

$$R = \begin{cases} \frac{\sum_{i=1}^{n}\left(\frac{t\_bid_i}{t\_actual_i}\right)}{n} & n \neq 0, \\ 1, & n = 0 \end{cases} \tag{12}$$

  In (12), assume that the service agent has executed some tasks for n times in the past. t_bid_i is the task execution duration as specified by the service agent in the bid for a given task, t_actual_i is the actual execution duration that

[3] The Critical Path algorithm is a graph algorithm which is used in project scheduling application. The description of this algorithm is outside the scope of this paper.

service agent spent on the execution of a given task each time. In case service agent cannot finish the task, t_actual_i is ∞. So, R is calculated based on bids in past negotiation sessions and task execution results in past workflow instances. These data are logged by the integration agent. It should be noted that the larger R is, the more reliable the service agent is. R will be used to estimate the service agent's actual execution duration $t\_estimate$ based on its current bid:

$$t\_estimate = \begin{cases} \frac{t\_bid}{R}, & R \neq 0, \\ \infty, & R = 0 \end{cases} \tag{13}$$

For example, assume that the service agent a has been assigned to execute the task t_i for five times. The corresponding bids and the execution results are listed in Table 1. In this example, $R \approx 0.9937 (R = \frac{\frac{20}{21}+\frac{15}{17}+\frac{17}{16}+\frac{15}{15}+\frac{15}{14}}{5})$. If

Table 1. Service agent a's bids and execution results

	Execution Time in Bid t_bid	Actual Execution Time t_actual
1	20 hours	21 hours
2	15 hours	17 hours
3	17 hours	16 hours
4	15 hours	15 hours
5	15 hours	14 hours

now service agent a indicates that it can finish the task in 20 hours in its bid, we can estimate that the service agent's actual execution duration is $\frac{20}{0.9937} \approx 20.13$ hours.

Based the reliability of service agents, total execution time of a plan p can be estimated using the function $t_e(p)$, where:

$$t_e(p) = CPT\left(t\_estimate_1, t\_estimate_2, ..., t\_estimate_m\right) \tag{14}$$

Selection Process. Execution plans are selected by the integration agent in two phases. In the first phase (pruning phase), execution plans are selected using a set of ECA (Event-Condition-Action) rules that are defined by process composers. These ECA rules use criteria which are defined above. For example, in (15), the ECA rule excludes the execution plans which have more than seven service agents, while in (16), the ECA rule excludes the execution plans with execution cost over 700 dollars.

$$\{Evaluate\_ExecutionPlan(p)\}[Community\_size(p) > 7]/Exclude(p) \tag{15}$$

$$\{Evaluate\_ExecutionPlan(p)\}[c(p) > 700 \text{ dollars}]/Exclude(p) \tag{16}$$

Assume that the set of execution plans P' is the result of the pruning phase. In the second phase (weighting phase), an execution plan will be selected from P'. The follows formula will be used:

$$V(p) = W_c \times \left(\frac{c(p)}{C_{planned}}\right) + W_t \times \left(\frac{t_e(p)}{T_{planned}}\right) + W_s \times \left(\frac{Z_p}{Z_w}\right) \qquad (17)$$

where, $C_{planned}$ (respectively, $T_{planned}$) represents the process composers expected total cost (respectively, work duration). Z_p is the size of the agent community, while Z_w is the number of tasks in the workflow. W_c,W_t and $W_s \in [0, 1]$, are the weights of cost, time and the size of agent community. Process composers are responsible for providing values of $C_{planned}$, $T_{planned}$, W_c, W_t and W_s.

In (17), process composers can balance cost, time and size of agent community to select a desired execution plan by adjusting the value of W_c, W_t and W_s. Specifically, if the cost is the most important factor, the weights can be set as follows: $W_c = 1$, $W_t = 0$, and $W_s = 0$. The integration agent will choose the execution plan which has the minimal value of $V(p)$. If there are more than one execution plans which have same minimal value of $V(p)$, then a plan will be selected randomly. We say that the selected one is an optimized execution plan.

4 Dynamic Business-to-Business Integration

In this section, we discuss how the integration agent coordinates with service agents to support dynamic B2B integration.

4.1 Execution Plan Evaluation

The integration agent starts the execution of the workflow instance based on the selected execution plan. It updates the execution plan based on the changes (e.g. a service agent fails when it is executing a task) which may happen during the execution of the workflow. The integration agent adopts a late binding strategy, which means it confirms the service agent's bid to execute task at the execution time. In this way, the integration agent can update the execution plan without withdrawing any accepted bid. Below, p_c is the current execution plan, p' contains the service agents which are executing workflow tasks, and T_u is the set of tasks which are waiting to be assigned to service agents.

$$p_c = \{< T_1^*, b_1, a_1 >, < T_2^*, b_2, a_2 >, ..., < T_x^*, b_x, a_x >\} \qquad (18)$$

$$p' = \{< T_j^*, b_j, a_j >, ..., < T_n^*, b_n, a_n >\} \qquad (19)$$

$$T_u = \{t_1, t_2, ..., t_i\} \qquad (20)$$

When a change happens, the integration agent considers the following cases:

1. **Case 1 (A service agent a fails when executing a task t after y unit of time):** In this case, the current execution plan becomes **un-executable**

and the task t needs to be re-executed. Thus, the integration agent needs to find another executable plan to continue the workflow execution. The integration agent reacts to the change by performing the following steps:-

- **Step 1.** Excludes all the execution plans which contain the service agent a's bids from P' (the result of pruning phase). The result is a set of execution plans P^*.
- **Step 2.** Selects execution plan p from P^*, such that:
  (a) $p = \{< T_1^*, b_1, a_1 >, < T_2^*, b_2, a_2 >, ..., < T_k^*, b_k, a_k >\}$,
  (b) $p' \subset p$, and
  (c) there exists some T_i^* in p's 3-tuples, and $\bigcup T_i^* = T_u \bigcup \{t\}$.
  P^{**} is the set of all the selected execution plan p in this step.
- **Step 3.** In any $p \in P^{**}$, for all the finished tasks, replaces the bids with task execution results. For the task t, add y to the execution time. P^{**} is used to select a new execution plan as discussed in Section 3.4.

Example 1. Figure 2. shows an example of case 1. In this example, a_2 fails so that it cannot finish t_3, a_5 and a_6 are executing their tasks. t_8, t_{11}, t_{12} and t_{13} are still waiting to be assigned to service agents. First, by excluding execution plans which contain a_2 from P', we have P^*. In P^*, select the execution plans in which a_5 and a_6 will execute same tasks as they do in the current execution plan and one service agent (i.e. a_1) executes only t_1.

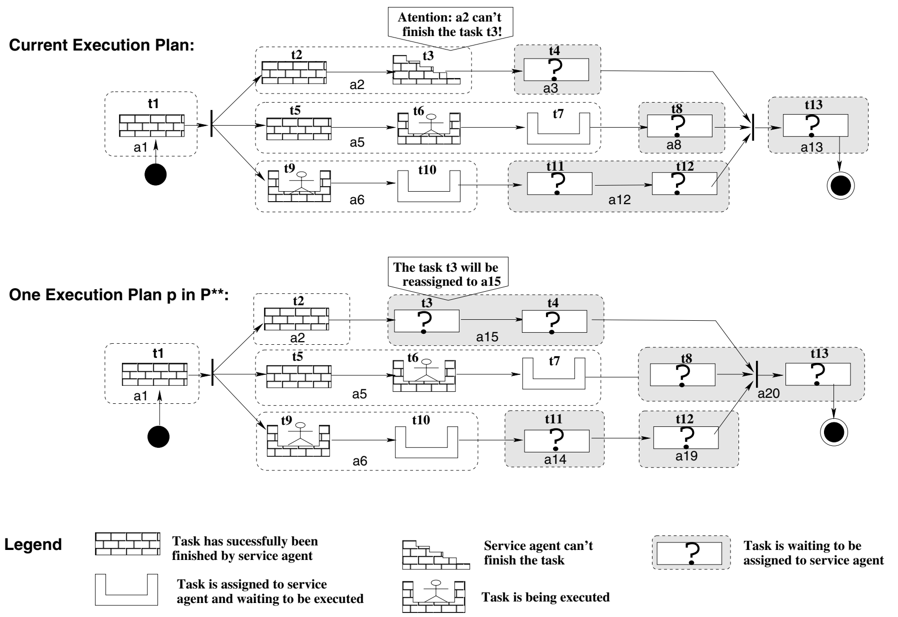

Current Execution Plan:
Atention: a2 can't finish the task t3!
t1
a1
t2
t3
a2
t4
a3
t5
t6
t7
a5
t8
a8
t13
a13
t9
t10
a6
t11
t12
a12
One Execution Plan p in P**:
The task t3 will be reassigned to a15
a15
a20
a14
a19
Legend
Task has sucessfully been finished by service agent
Service agent can't finish the task
Task is waiting to be assigned to service agent
Task is assigned to service agent and waiting to be executed
Task is being executed

Fig. 2. Example 1: The service agent a_2 fails after it had been executing the task t_3 for 5 hours

The p in Figure 2 is an example of such execution plan. Note that in p, t_3 will be re-assigned to a_{15}. For the finished tasks t_1, t_2 and t_5, the bids will be replaced by the task execution results. So, the integration agent can use p to continue the execution of the workflow.

2. **Case 2 (A service agent a has finished the execution of the tasks T_l^*, but the execution duration is different from what the integration agent estimated):** In this case, the current execution plan is still executable. But this change may make another execution plan become a better alternative. So the integration agent reacts to the change by performing the following steps:-

   – **Step 1.** Selects execution plan p from P', such that:
   (a) $p = \{< T_1^*, b_1, a_1 >, < T_2^*, b_2, a_2 >, ..., < T_k^*, b_k, a_k >\}$,
   (b) $p' \subset p$, and
   (c) there exists some T_i^* in p's 3-tuples, and $\bigcup T_i^* = T_u$.
   P^{**} is the set of all the selected execution plan p in this step.
   – **Step 2.** In any $p \in P^{**}$, for the finished tasks, replaces the bids with execution results. Then P^{**} is used to select a plan p_t as discussed in Section 3.4. If value of $V(p_t)$ is less than value of $V(p_c)$, integration agent will change the current execution plan to p_t. Otherwise, integration agent keeps the p_c to execute the workflow.

Example 2. Figure 3 shows an example of case 2. In this example, when a_5 is executing t_7, a_2 has finished t_3 earlier than that the integration agent

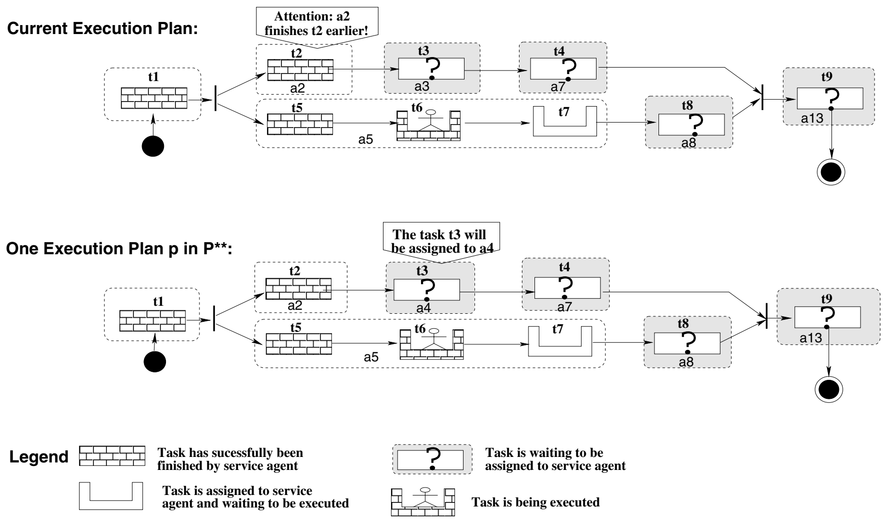

Current Execution Plan:
Attention: a2 finishes t2 earlier!
t1
t2
a2
t3
a3
t4
a7
t5
t6
a5
t7
t8
a8
t9
a13
One Execution Plan p in P**:
The task t3 will be assigned to a4
t1
t2
a2
t3
a4
t4
a7
t5
t6
a5
t7
t8
a8
t9
a13
Legend
Task has sucessfully been finished by service agent
Task is waiting to be assigned to service agent
Task is assigned to service agent and waiting to be executed
Task is being executed

Fig. 3. Example 2: The service agent a_2 finish execution the task t_2 earlier than the integration agent estimated

estimated. So, t_4 can have more time to be executed. The integration agent can select a_4 to execute t_4 if a_4 asks for less execution cost comparing to a_3.

3. **Case 3 (A service agent a sends a new bid b for the tasks T_i^*, and T_i^* is a subset of T_u):** The bid b can be used to generate a better alternative execution plan than the current execution plan. So the integration agent reacts to the change by performing the following steps:

   – **Step 1.** Selects execution plan p in P', if:
   (a) $p = \{< T_1^*, b_1, a_1 >, < T_2^*, b_2, a_2 >, ..., < T_k^*, b_k, a_k >\}$,
   (b) there exists some T_i^* in p's 3-tuples, and $\bigcup T_i^* = T_u$,
   (c) $p' \subset p$, and
   (d) there exists some T_j^* in p's 3-tuples, and $\bigcup T_j^* = T^*$
   P^* is the set of all the selected execution plans p in this step.
   – **Step 2.** For each execution p in P^*, replace all the 3-tuples $< T_j^*, b_j, a_j >$ with the 3-tuple $< T^*, b, a >$, generates a new execution plans set P^{**}.
   – **Step 3.** In any $p \in P^{**}$, for the finished tasks, replaces the bids with task execution results. Then P^{**} is used to select a plan p_t as discussed in Section 3.4. If the value $V(p_t)$ is less than value of $V(p_c)$, integration agent will change the current execution plan to p_t.

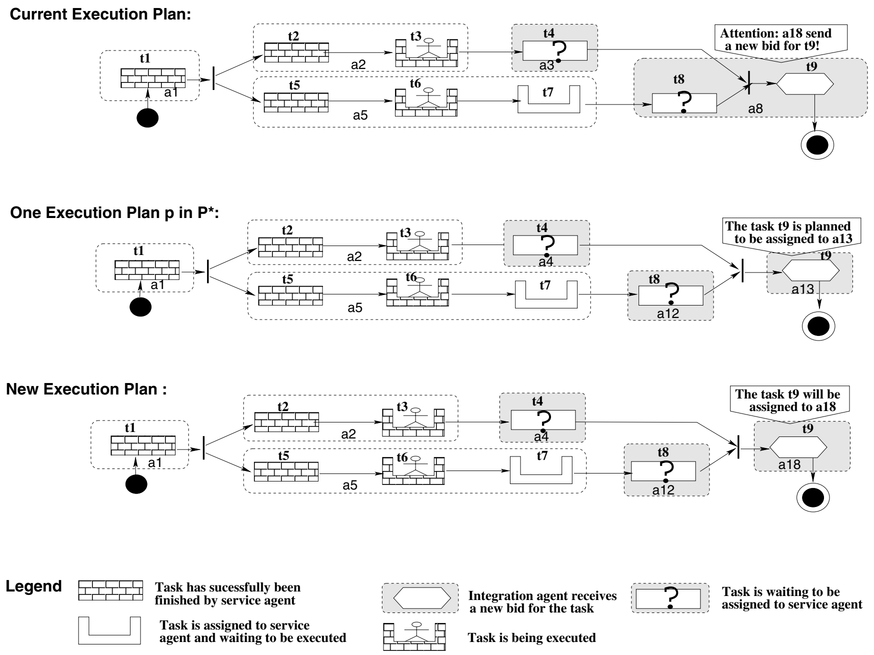

Current Execution Plan:
t1
a1
t2
a2
t3
t4
a3
t5
a5
t6
t7
t8
a8
t9
Attention: a18 send a new bid for t9!
One Execution Plan p in P*:
a4
a12
a13
The task t9 is planned to be assigned to a13
New Execution Plan :
a18
The task t9 will be assigned to a18
Legend
Task has sucessfully been finished by service agent
Task is assigned to service agent and waiting to be executed
Integration agent receives a new bid for the task
Task is being executed
Task is waiting to be assigned to service agent

Fig. 4. Case 3: The service agent a_{18} sends a new bid b for the task t_9

Example 3. Figure 4 shows an example of case 3. In this example, when a_2 and a_5 are currently executing tasks (i.e. t_3 and t_6), a_{18} sends a new bid for t_9. The integration agent searches the execution plan p where a_2 and a_5 can execute same tasks as they do in current execution plan, one service agent (i.e. a_1) can execute only t_1 and the other one service agent (i.e. a_{13}) can execute only t_9. p is such an execution plan. New execution plan can be created by replacing a_{13} with a_{18}.

4.2 Workflow Execution

Based on a selected execution plan, the integration agent composes the service agents to execute the workflow. During the execution of the workflow, the integration agent acts as a facilitator for service agents to exchange information. Meanwhile, the integration agent acts as a monitor. It periodically pings the service agents when they are executing tasks, in order to detect whether any of them have failed or not. If the service agent has exhausted the estimated execution time and still cannot finish executing the allocated tasks, the integration agent will ping it more frequently, because such service agent is more prone to fail. Service agents also send messages to the integration agent when they have finished certain action during the execution of the task. The integration agent forwards such messages to the user agent so that users can be aware of the progress of workflow execution. The integration agent also logs the execution results of each task, which will be useful for selecting execution plans for other workflow instance. The agent community will dismiss after the workflow has been finished. A new agent community will be formed for next workflow instance.

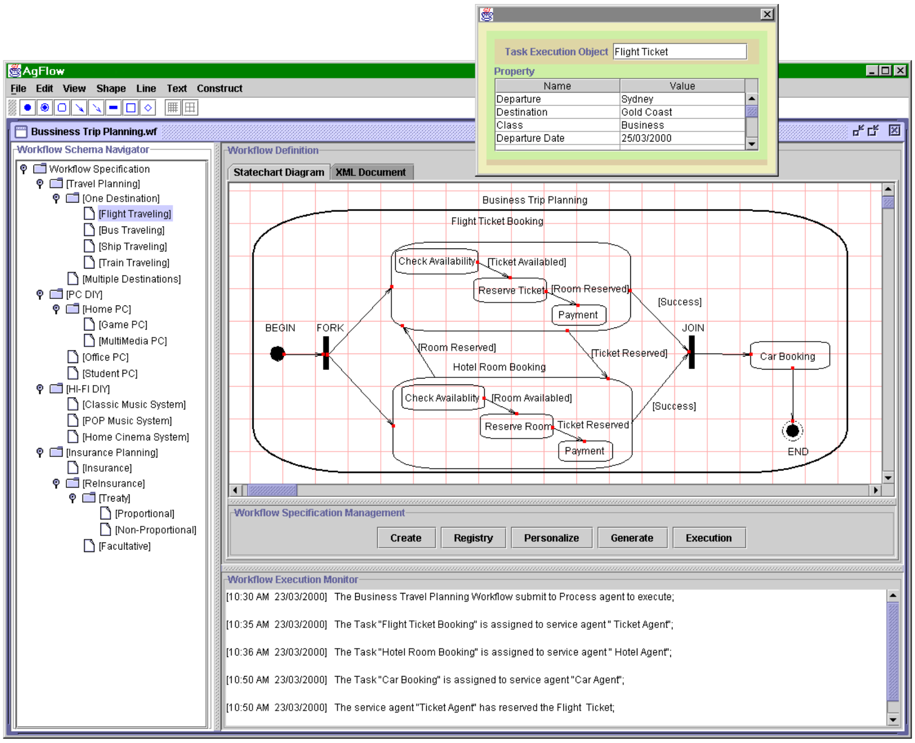
5 Implementation

The AgFlow architecture is shown in Figure 1. The prototype is developed using EJB (*Enterprise JavaBeans*). Agents communicate through a communication bus which is implemented using JSDT (Java Shared Data Toolkit). The user agent provides a GUI (Graphical User Interface) to access the AgFlow system. It is implemented using Java. Currently, three types of services are considered in the AgFlow: Domino-based workflows, Enterprise JavaBean applications, and Java applications that allow access to relational databases via JDBC. We are currently working on wrapping other types of services including Oracle Workflow and SAP/R3.

We consider a business trip planning workflow to illustrate the use of the AgFlow. The workflow schema includes three tasks which are flight ticket booking (i.e. t_1), hotel room booking (i.e. t_2) and car booking (i.e. t_3). The workflow has the following constraints:

– The salesman can book a flight ticket and a hotel room at the same time.
– If both bookings are successful, there is the third task: car booking.

We developed 14 service agents ($a_1, a_2, ..., a_{14}$) that wrap three types of services (Domino-based workflows, Enterprise JavaBean and JDBC applications). All the service agents are registered with the discovery agent. We will show how to define a workflow schema, create a workflow instance, and execute it using the AgFlow.



Task Execution Object: Flight Ticket

Property

Name	Value
Departure	Sydney
Destination	Gold Coast
Class	Business
Departure Date	25/03/2000

```
[10:30 AM  23/03/2000]  The Business Travel Planning Workflow submit to Process agent to execute;

[10:35 AM  23/03/2000]  The Task "Flight Ticket Booking" is assigned to service agent " Ticket Agent";

[10:36 AM  23/03/2000]  The Task "Hotel Room Booking" is assigned to service agent " Hotel Agent";

[10:50 AM  23/03/2000]  The Task "Car Booking" is assigned to service agent "Car Agent";

[10:50 AM  23/03/2000]  The service agent "Ticket Agent" has reserved the Flight  Ticket;
```

Fig. 5. User Agent

1. **Workflow Schemas.** Workflow schemas are organized using domain-specific hierarchy. In left panel, every non-leaf node in the hierarchy represents a set of workflow schemas of a specific domain. Every leaf node represents a schema of particular workflow. For example, in Workflow Schema Navigator panel of Figure 5, there are four workflow domains namely, `Travel planning, PC DIY, HI-FI DIY and Insurance planning`. `Home PC` is sub-domain of `PC DIY` and has two workflow schemas (i.e., `Game PC and Multimedia PC`). The schema of the business trip planning workflow is defined as shown in the draw panel of Figure 5. This schema is stored as a leaf node called `Business Trip` of the domain `One Destination`.
2. **Workflow Instances.** An end user can browse workflow schemas using the Workflow Schema Navigator panel. When she/he clicks on a leaf node, the

corresponding UML statechart is displayed in Workflow Definition panel. An instance of the workflow is created by clicking on the `Create` button in the Workflow Management panel. The user can then supply values of the parameters that are needed to execute the workflow (e.g., the destination and departure date). After that, the user can click on the `Generate` button to generate the XML document that describes the workflow. After clicking on the `Execution` button, the XML document is sent to the integration agent to execute the workflow.

3. **Dynamic Business-to-Business Integration.** The integration agent parses the XML document into three tasks. For each task, it queries the discovery agent to locate the relevant service agents. C^* is created by the integration agent based on query results. $C^* = \{< A'_1, T_1 >, < A'_2, T_2 >, ..., < A'_5, T_5 >\}$, $A'_1 = \{a_1, a_2, a_3\}$, $T_1 = \{t_1\}$, $A'_2 = \{a_4, a_5, a_6\}$, $T_2 = \{t_2\}$, $A'_3 = \{a_7, a_8, \}$, $T_3 = \{t_3\}$, $A'_4 = \{a_9, a_{10}, a_{11}\}$, $T_4 = \{t_1, t_2\}$, $A'_5 = \{a_{12}, a_{13}, a_{14}\}$, $T_5 = \{t_1, t_3\}$. Since there are five 2-tuples in C^*, the integration agent creates five negotiation sessions to negotiate with service agents. The set of execution schemas is: $S = \{s_1, s_2, s_3\}$ where $s_1 = \{T_1, T_2, T_3\}$, $s_2 = \{T_3, T_4\}$ and $s_3 = \{T_2, T_5\}$. Based on the available bids in negotiation sessions, the integration agent will create a set of execution plans. An execution plan will be selected as discussed in Section 3.4. Assume that the selected execution plan is $p = \{< T_2, b_2, a_5 >, < T_5, b_5, a_{12} >\}$. Based on the execution plan p, the integration agent will integrate service agent a_5 and a_{12} to execute the business trip planning workflow.

6 Related Work and Conclusions

Currently, the use of the cross-enterprise workflow is beginning to gather momentum in B2B integration. Related projects include the CMI [4], CrossFlow [3] and eFlow [1]. In CMI, a Service Oriented Process model is proposed. This model provides a set of service management primitives such as `activity place holder` and `dynamic role assignment`. However, CMI has not addressed the issue of brokering and selection of services that goes beyond what is stated in the service interfaces. CrossFlow and eFlow focus on loosely coupled processes. They consider some important requirements of B2B such as adaptability and external manageability. However, the dynamic integration of service is not explicitly supported.

As discussed in Section 1, agent technology is one of the promising candidates for E-commerce environment. Several types of agents have been used in E-commerce applications, including source, discovery and mediator agents. The use of software agents in automating a single organization business processes have been discussed in several publications [2,6,7]. In [2], each workflow is represented by multiple personal agents, actor agents and authorization agents. These agents act as personal assistants performing actions on behalf of the workflow participants and facilitating interaction with other participants or organization specific WFMS. In [6], the multi-agent architecture consists of a number of autonomous agencies. A single agency consists of a set of subsidiary agencies which is controlled by one responsible agent. Each agent is able to perform one or more services. These atomic agents can be combined to form complex services by adding ordering constraints and conditional control. However, neither [2] nor [6] adopts agent technology to compose workflow execution dynamically. In [7], a multi-plan state machine agent model is presented. Agents are assembled dynamically from the descriptions in a blueprint language and can be modified while running. But users have to describe in detail how agents can be assembled together and what changes are allowed in apriori.

In this paper, we have presented the design and implementation of the AgFlow, an agent-based workflow system for dynamic business-to-business integration. In our approach, the agent community for executing a specific workflow instance is optimally and automatically composed based on the context of workflow execution. It can self-adapt and react to changes during execution.

References

1. Fabio Casati, Ski Ilnicki, Li-Jie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. eFlow: a Platform for Developing and Managing Composite e-Services. Technical Report HPL-2000-36, HP Laboratoris Palo Alto, 2000. 416
2. J. Chang and C. Scott. Agent-based workflow: TRP support Environment (TSE). *Computer Networks and ISDN Systems*, 28(7-11):1501–1511, 1996. 416, 417
3. CrossFlow project web page, 2000. http://www.crossflow.org. 416
4. D. Georgakopoulos, H. Schuster, A. Cichocki, and D. Baker. Managing process and service fusion in virtual enterprises. *Information System, Special Issue on Information System Support for Electronic Commerce*, 24(6):429–456, 1999. 416
5. M. Huhns and M. Singh, editors. *Internet-Based Agents*, IEEE Internet Computing. IEEE, July 1997. Special Issue on Agents. 403
6. N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. O'Brien, and M. E. Wiegand. Agent-based business process management. *International Journal of Cooperative Information Systems*, 5(2&3):105–130, 1996. 416, 417
7. Krzysztof Palacz and Dan C. Marinescu. An agent-based workflow management system. Technical report, Computer Sciences Department, Purdue University, 1999. 416, 417
8. Steven S. Skiena. *The Algorithm Design Manual.* Springer Verlag, 1997. 407
9. Jeffrey D. Smith. *Design and Analysis of Algorithms.* PWS-KENT Publishing Company, 1989. 408
10. Liangzhao Zeng, Boualem Benatallah, and Anne Hee Hiong Ngu. Agent-based service discovery for workflow integration. In *Proceeding of the 2001 International Symposium on Information Systems and Engineering*, Las Vegas, Nevada, USA, 2001. 405
11. Liangzhao Zeng, Boualem Benatallah, Anne Hee Hiong Ngu, and Phuong Nguyen. Agflow: Agent-based cross-enterprise workflow management system(demonstration paper). In *Proceedings of 27th International Conference on Very Large Data Bases*, Roma, Italy, 2001. 404

Yoda: An Accurate and Scalable Web-Based Recommendation System

Cyrus Shahabi, Farnoush Banaei-Kashani, Yi-Shin Chen, and Dennis McLeod

Department of Computer Science, Integrated Media Systems Center
University of Southern California, Los Angeles, CA 90089-2561, USA
{shahabi,banaeika,yishinc,mcleod}@usc.edu

Abstract. Recommendation systems are applied to personalize and customize the Web environment. We have developed a recommendation system, termed *Yoda*, that is designed to support large-scale Web-based applications requiring highly accurate recommendations in real-time. With Yoda, we introduce a hybrid approach that combines collaborative filtering (CF) and content-based querying to achieve higher accuracy. Yoda is structured as a tunable model that is trained off-line and employed for real-time recommendation on-line. The on-line process benefits from an optimized aggregation function with low complexity that allows real-time weighted aggregation of the soft classification of active users to pre-defined recommendation sets. Leveraging on *localized* distribution of the recommendable items, the same aggregation function is further optimized for the off-line process to reduce the time complexity of constructing the pre-defined recommendation sets of the model. To make the off-line process scalable furthermore, we also propose a filtering mechanism, *FLSH*, that extends the *Locality Sensitive Hashing* technique by incorporating a novel distance measure that satisfies specific requirements of our application. Our end-to-end experiments show while Yoda's complexity is low and remains constant as the number of users and/or items grow, its accuracy surpasses that of the basic nearest-neighbor method by a wide margin (in most cases more than 100%).

1 Introduction

The World Wide Web is emerging as an appropriate environment for business transactions and user-organization interactions, because it is convenient, fast, and cheap to use. Witness the enormous popularity of e-Commerce and e-Government applications. However, since the Web is a collection of semi-structured and structured information resources, Web users often suffer from information overload. To remedy this problem, recommendation systems are appropriate to personalize and customize the Web environment. Mass customization of the Web facilitates access to the Web-based information resources and realizes the full economic potential of the Web. Recommendation systems have achieved widespread success in e-Commerce and e-Government applications.

Various statistical and knowledge discovery techniques have been proposed and applied for recommendation systems. To date, *collaborative filtering (CF)*

© Springer-Verlag Berlin Heidelberg 2001

is the most successful approach employed in recommendation systems [5,9]; it is used in many of the real recommendation systems on the Web. Collaborative filtering works based on the assumption that if user x interests are similar to user(s) y interests, the items preferred by y can be recommended to x. With collaborative filtering, the system strives to predict unknown interests of the user based on similarity of the user interests with those of other users.

There are two fundamental challenges for CF-based recommendation systems. The first challenge is to improve the scalability of the system. For modern e-Commerce applications such as eBay.com™ and Amazon.com™, a recommendation system should be able to provide recommendations in real-time while the number of both users and items exceed millions. There are two families of collaborative filtering algorithms: *memory-based* algorithms [1,5], and *model-based* algorithms [8,9]. With memory-based algoritms, the entire recommendation process is generally an on-line process. On the other hand, with model-based algorithms recommendation is performed using an aggregated model. Thus, the time-consuming part of the recommendation process is performed off-line, leaving the on-line process with reasonably low time complexity. Hence, model-based approaches are preferred in large-scale applications, where millions of recommendations are to be generated in real-time. The second challenge for CF-based recommendation systems is to improve the accuracy of the recommendations to be as efficacious as possible. False recommendations, such as those that miss recommendable items (termed *false negatives*), or those that include non-recommendable items (termed *false positives*), seriously affect efficacy of the recommendation systems and must be avoided. Problems such as *sparsity* and *synonymy* further complicate enhancement of the accuracy (see Section 2). Accuracy is usually sacrificed to achieve lower space and time complexity.

In this paper, we present our scalable and accurate CF-based recommendation system, termed *Yoda*, which is designed to support large-scale Web-based applications requiring highly accurate recommendations in real-time. With Yoda, we introduce a tunable model-based approach that can strike a compromise between scalability and accuracy based on the specific application requirements. With our model, not only we optimize the *on-line* recommendation process, but also we propose techniques to reduce time complexity of generating the model during the *off-line* process. During the online process, observing the probable multi-nature behavior of each single user, first we *softly* classify an active user based on typical patterns of users behaviors. Subsequently, we perform a weighted aggregation on *pre-defined* recommendations associated to each class of behaviors to generate the *soft* recommendations for the user. Applying an accurate and tunable aggregated model, *FM*, and a customized similarity measure, *PPED*, accurate classification of users is performed in real-time [13]. To expedite the aggregation step, we propose an optimized fuzzy aggregation function that reduces time complexity of the aggregation from $O(\|I\| \times \|P\|)$ to $O(\|I\|)$ (where I and P are the size of the item set and size of the property set associated with each item, respectively).

On the other hand, with large-scale applications since both items presented within the Web-site and user behaviors are changing rapidly, the model itself must be updated/regenerated frequently, e.g. once a day. We apply the same aggregation function to generate the pre-defined recommendations, *cluster wish-lists*, for each class of users during the off-line process. In this case, we view each cluster wish-list as a sub-system. Hence, we can incorporate an optimized aggregation algorithm proposed by Fagin [2] to further reduce the time complexity of the aggregation function to $O(\|N\|)$, where N is a small constant parameter selected during system tuning. Furthermore, we take advantage of the localized distribution of the items and propose an extended version of the *LSH* technique [15], termed *FLSH*, to avoid considering the items that do not show sufficient proximity to the required recommendations, while generating the cluster wish-lists. The original LSH is appropriate for fast item retrieval (sublinear to $\|I\|$) in high-dimensional spaces (large $\|P\|$), which makes it the optimum choice for cluster wish-list generation. However, the distance measure used in LSH, Hamming distance, does not satisfy requirements of our distance space. With FLSH, we introduce and incorporate our own distance measure to customize LSH for our application.

In sum, the major contribution of this paper is a novel content-based ranking method for CF-based models that captures associations between items. Consequently, our model considers both association between users - with collaborative filtering and items - with content-based filtering; this is to address the synonym problem of the pure CF-based approaches, and achieve higher accuracy even very large-scale applications.

2 Related Work

Recommendation systems are designed either based on *content-based filtering* or *collaborative filtering*. Content-based filtering approaches are derived from the concepts introduced by the Information Retrieval (IR) community. Content-based filtering systems are usually criticized for two weaknesses, **content limitation** (e.g., IR methods can only be applied to a few kinds of content) and **over-specialization**. The collaborative filtering (CF) approach remedies these two problems. Typically, CF-based recommendation systems do not use the actual content of the items to evaluate them for recommendation. Moreover, since other user profiles are also considered, user can explore new items which are in terms with avoiding the over-specialization problem. The nearest-neighbor algorithm is the earliest CF-based algorithm used in recommendation systems [5]. With this algorithm, the similarity between users is evaluated based on their rating data, and the recommendation is generated considering the items visited by nearest neighbors of the user. In its original form, CF-base recommendations suffer from the problems of, **scalability**, **sparsity**, and **synonymy** (i.e., latent association between items is not considered for recommendations.)

In order to alleviate or even eliminate these problems, more recently, researchers introduced a variety of different techniques into collaborative filtering systems, such as *content analysis* [4] for avoiding the synonymy and sparsity problems, *categorization* [7] to alleviate the synonymy and sparsity problems, *bayesian network* [9,8] for lightening the scalability problem, *clustering* [9] to lessen sparsity and scalability problems, and Singular Value Decomposition (SVD) [6,10] to ease all three problems to a certain limit.

With Yoda, we introduce an integrated model which brings together the advantages of *model based*, *clustering*, *content analysis*, and *CF* approaches. As a result, we reduce the time complexity through model based and clustering approaches, alleviate the synonymy problem with content analysis method, and address the sparsity problem by implicit identification of the users interests [11,12].

3 Overview

The objective of a Web-based recommendation system can be stated as follows:

Problem Statement Suppose $I = \{i | \text{“}i\text{” } \textit{is an item}\}$ is the set of items presented in a Web-site, termed the *item-set* of the Web-site, and x is a user interactively navigating the Web-site. The recommendation problem is defined to find a ranked list of the items I_x, termed *wish-list*, in which items in I_x are ranked based on x interests.

To provide a wish-list for a user, generally a CF-based recommendation system goes through 2 steps/phases:

1. User Classification: During this phase, data about user interests are acquired and employed to classify the user.
2. Ranking the Items: In this phase, the predicted user interests are applied to rank and order the items in the item-set to provide the final wish-list for the user.

To illustrate the processing flow of Yoda, consider Figure 1. Suppose music CDs are the items presented by a given web-site. For a music CD, for instance proximity to various styles of music such as Classic, Rock, Pop, etc. can be considered as different properties of the item. Yoda is to provide each active user of the site with purchase recommendations that are compatible with the style(s) of music the user likes. To generate the recommendations, during an off-line process Yoda uses fuzzy terms such as *Low*, *Medium*, and *High* to evaluate correspondence of each CD with a defined set of properties, e.g. {*Rock*, *Pop*, *Indie*, *Heavy-Metal*, *Jazz*}. For example, a CD can be labeled with *Rock* = *High*, *Pop* = *Low*, *Indie* = *Low*, *Heavy-Metal*= *Medium*, and *Jazz* = *Low*. Also, during the off-line process, Yoda identifies similar groups of users by clustering user sessions from a training set, and learns typical pattern of users interests in each cluster, termed *favorite property values (favorite PVs)* of the cluster, by taking vote among items browsed by users belonging to the cluster. For instance, favorite PVs of a cluster can be *Rock* = *High*, *Pop* = *Low*, *Indie* =

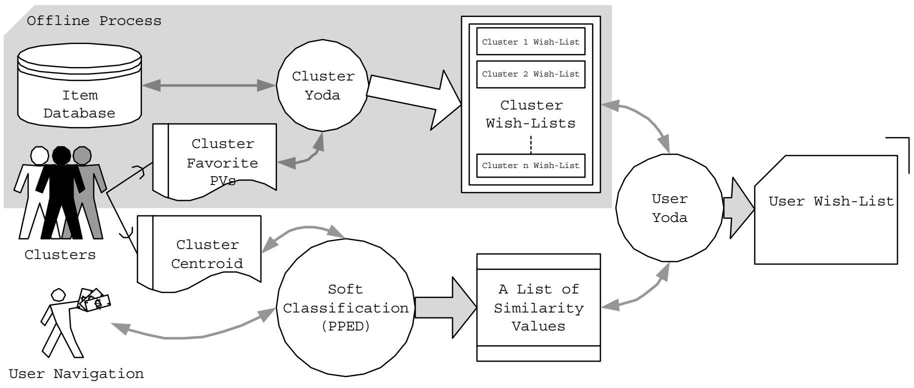


Fig. 1. Architecture of Yoda

Medium, *Heavy-Metal*= *High*, and *Jazz* = *Low*. Then, Yoda applies favorite PVs of each cluster as a measure to rank items in the item-set and predict a list of recommendations, termed *wish-list*, for each cluster. Later, during the online recommendation process first an active user is softly classified with clusters of users based on his/her partial navigation pattern. Thereafter, Yoda uses the classification factors to generate the final wish-list for the active user by weighted aggregation of the cluster wish-lists. Items in the user wish-list are now ranked based on preferences of the corresponding user.

4 System Design

In this section, we provide a detailed description of Yoda's components. Since phase I is performed based on our previous work [12,13], here we elaborate more on phase II of Yoda.

4.1 Phase I - User Classification

Yoda uses the client-side tracking mechanism we proposed in [12] to capture view-time, hit-count, and sequence of visiting the web-pages that particularly provide information about the items presented within a web-site. These features are applied to infer users interests in items. To analyze these features and infer the user interests, Yoda employs a flexible and accurate model we introduced in [13], the *Feature Matrices (FM)* model. FM is a set of variable-order hypercube data structures that can represent various aggregated access features with any required precision. With FM, we can model access patterns of both a single user, and a cluster of users. Here, Yoda uses FM to model the access patterns of the active users individually and the aggregated access pattern of each cluster of users from the training data. Yoda also applies a version of the similarity measure we proposed in [13], *Projected Pure Euclidean Distance (PPED)*, to evaluate the similarity of a user access/interest pattern to a cluster access/interest pattern modeled by FM. PPED allows highly accurate classification of the users' partial interest patterns. Since essentially PPED is a *dissimilarity* measure, here we use a slightly different version of PPED to quantify *similarity* of user u to cluster k, S_{uk}:

$$S_{uk} \;=\; MaxDistance \;-\; TPPED(u,k) \qquad 0 \le S_{uk} \le MaxDistance$$

where $MaxDistance$ is a constant parameter that is selected as the upper bound for distance, and Truncated PPED (TPPED) is defined as follows:

$$TPPED(u,k) = \begin{cases} PPED(u,k) & if\ PPED(u,k) \le MaxDistance \\ MaxDistance & if\ PPED(u,k) > MaxDistance \end{cases}$$

Thus, by computing S_{uk_i}, Yoda can quantify the similarity of user u interests to the interest pattern of each cluster k_i and *softly* classify user interests to typical interest patterns within the web-site. During the item ranking process (see Section 4.2), Yoda uses S_{uk_i} as the the weight factor for aggregating wish-list of cluster k_i into the final user wish-list.

4.2 Phase II - Ranking the Items

In this section, first we formally define some terms. Second, we explain the off-line process through which the cluster wish-lists are constructed. Third, we describe how user wish-list is generated during the on-line process. Finally, we explain a filtering mechanism to optimize the time complexity of generating cluster wish-lists.

Defining Terms Here, we define the notions of property, item, and user/cluster wish-list. An *item* is an instance of product, service, etc. that is presented in a web-site. The set of items presented in a web-site comprise the *item-set*, I, of the web-site. Items are described by their *properties*, which are abstract perceptual features. For example, for a music CD as an item, "styles" of the music (such as Classic, Rock and Pop) and "ratings" of this CD by different critics can be considered as properties of the item. Since properties are perceptual we use fuzzy-sets to evaluate properties [14].

Definition If $\varphi = \{f_l \mid f_l$ is a fuzzy set, and $\forall l \in N - \{1\} \quad f_l > f_{l-1}\}$ and $P = \{p \mid p$ is a property$\}$, then an item $i \in I$ is defined as:

$$i = \{(p, \tilde{p}_i) \mid p \in P, \tilde{p}_i \in \varphi\}$$

Example Suppose item K is defined as: { (Pop, high), (Rock, low), (Critic-A, good), (Critic-C, neutral)}. It represents that the style of this item is very similar to "Pop" and only a little similar to "Rock". Moreover, it also describes the opinions of two critics.

Definition A wish-list, I_x, for user/cluster x is defined as:

$$I_x = \{(i, v_x(i)) | i \in I, v_x(i) \in [0, 1]\}$$

where the *preference* value $v_x(i)$ measures the probability of x being interested in the item i.

Generating Cluster Wish-Lists (Off-Line Process) Yoda represents the aggregated interests of the users in each cluster by a set of property values (PVs), termed *favorite PVs* of the cluster. Each favorite PV identifies likelihood of the cluster being interested in a specific property of the items. Favorite PVs of each cluster are extracted by applying a *voting procedure* to the set of items visited by the users of a cluster, termed the *browse-list* of the cluster, as follows:

Definition *User browse-list*, b_u, and *cluster browse-list*, B_k, are defined as:

$$b_u = \{i \mid i \in I, \text{"}i\text{" is visited by } u \in U\}$$
$$B_k = \bigcup_{u \in k} b_u$$

where U is the training set of users. The voting procedure extracts the favorite PV, $F_p(k)$, corresponding to property p for cluster k as follows[1]:

$$C_{p,f}(k) = \|\{i \mid i \in B_k, \tilde{p}_i = f\}\|$$
$$F_p(k) = fmax\{f \mid f \in \varphi, C_{p,f}(k) = \max_{\forall f' \in \varphi} \{C_{p,f'}(k)\}\}$$

Example Suppose the browse-list of cluster K is {CD-A, CD-B, CD-G, CD-K, CD-Y, CD-Z}, and the values of property "Rock" for the corresponding CDs are { (CD-A, high), (CD-B, high), (CD-G, low), (CD-K, medium), (CD-Y, high), (CD-Z, high) }. Because "high" has the maximum vote, the favorite PV of cluster K, $F_{Rock}(K)$, is "high".

Cluster-Yoda is the module that evaluates $v_k(i)$, preference value of an item i for cluster k. To compute $v_k(i)$, cluster-Yoda employs a fuzzy aggregation function to measure and quantify the similarity between favorite PVs of the cluster k and specific property values associated with the item i. We use an optimized aggregation function with a triangular norm, which satisfies *conservation*, *monotonicity*, *commutativity*, and *associativity* requirements for data aggregation [2]. Here, we formally define the aggregation function used to compute $v_k(i)$:

Definition First, properties are grouped based on their corresponding values in favorite PVs of the cluster k:

$$G_f(k) = \{p \mid f \in \varphi,\ p \in P,\ F_p(k) = f\}$$

[1] "$fmax$" is the fuzzy max function.

then, the preference value $v_k(i)$ for item i is computed as:

$$E_{k,f}(i) = f \times fmax\{\tilde{p}_i \mid p \in G_f(k)\}$$
$$v_k(i) = fmax\{E_{k,f}(i) \mid \forall f \in \varphi\}$$

Example Suppose properties are grouped as $G_{medium}(K)$ = {Vocal, Soundtrack}, $G_{high}(K)$ ={Rock, Pop}, and $G_{low}(K)$ ={Classic}, and the item i is defined as { (Rock, low), (Pop, low), (Vocal, low), (Soundtrack, high), (Classic, medium)}. According to the equations above, the preference value $v_K(i) = fmax$ { (high $\times$ low), (medium $\times$ high), (low $\times$ low) } = (medium $\times$ high) = 0.75.

Basically, this aggregation function partitions the properties into $\|\varphi\|$ different subgroups according to the favorite PVs of the cluster k. Subsequently, the system maintains a list of maximum property values for all subgroups. Finally, the system computes the preferences of all items in the cluster wish-list by iterating through all subgroups. As compared to a naive weighted aggregation function with time complexity $O(\|P\| \times \|I\|)$, the complexity of the proposed aggregation function is $O(\|\varphi\| \times \|I\|) = O(\|I\|)$, where $\|\varphi\|$ is a small constant number.

To reduce the time complexity of generating the cluster wish-lists further, we apply a cut-off point to the cluster wish-lists. Each cut wish-list includes the N best-ranked items according to their preference values for the corresponding cluster. In [2], Fagin has proposed an optimized algorithm, the A_0 algorithm, to retrieve N best items from a collection of subsets of items with time complexity proportional to N rather than total number of items. Here, by taking the subgroups of items (as described above) as the subsets, the A_0 algorithm can be incorporated into the cluster-Yoda[2]. Applying the A_0 algorithm to generate a cluster wish-list with cut-off point N, we reduce the time complexity to $O(\|\varphi\| \times \|N\|) = O(\|N\|)$, where $\|N\| \ll \|I\|$.

Generating User Wish-Lists (On-Line Process) During the on-line recommendation process, *user-Yoda*, which is an aggregation module similar to the cluster-Yoda, aggregates the cluster wish-lists to generate the final resolved user wish-list for the active user u. User-Yoda applies a fuzzy aggregation function to compute the preference value $v_u(i)$ of each item i ($i \in \bigcup_{\forall k} I_k$) for the user u based on similarity S_{uk} of user u to clusters k (where $i \in I_k$).

Definition First, clusters are grouped based on their similarity to the user u as follows[3]:

[2] Since our aggregation function is in triangular norm form, it satisfies the requirements of the A_0 algorithm.

[3] Numerical S_{uk} values are converted to fuzzy equivalents. See [14] for details of the conversion procedure.

$$G_f(u) = \{k \mid f \in \varphi,\ S_{uk} = f\}$$

then, the preference value $v_u(i)$ for item i is computed as:

$$E_{u,f}(i) = f \times fmax\{v_i(k) \mid k \in G_f(u)\}$$
$$v_i(u) = fmax\{E_{u,f}(i) \mid \forall f \in \varphi\}$$

Filtering Mechanism Cluster-Yoda generates a cluster wish-list based on the favorite PVs of the cluster. The large size of the item-set renders our approach to off-line ranking practically time complex. Therefore, we have to incorporate a mechanism into the cluster-Yoda to target the set of items that more probably contribute towards higher preference values.

The item-set in large-scale applications is a high-dimensional database. In [16], it is demonstrated that all index structures degrade to a linear search for sufficiently high dimensions. On the other hand, we observe that items comprising the item-set are *locally* distributed; a property that can be exploited to reduce time complexity of the item selection for the cut cluster wish-lists. We extend the *Locality Sensitive Hashing (LSH)* algorithm proposed in [15] to hash items into hash buckets preserving the locality in storage. This algorithm has sublinear dependency on the item-set size, even when the item-set is a high-dimensional database. However, a fuzzy feature space bears the property that the higher fuzzy terms (i.e. $f_{l_1} > f_{l_2}$, then f_{l_1} is *higher*) show more proximity to the solution space as opposed to the lower fuzzy terms. Thus, we have developed a novel fuzzy distance measure as opposed to the Hamming distance measure used by [15] to consider this property. We term the locality sensitive hashing algorithm customized for our application as *Fuzzy Locality Sensitive Hashing (FLSH)*. FLSH reduces the *potential* solution space of the problem stated below, so that the aggregation function of the cluster. Yoda achieves the ideal solution with lower time complexity:

Problem Statement Find the N nearest neighbors for favorite PVs of cluster k, V_k:

$$V_k = \{(p, F_p(k)) \mid p \epsilon P\}$$

from the item-set I, where $\|I\| = M \gg N$.

First, Yoda hashes the items in the item-set I using the LSH algorithm. Each item $i \in I$ is embedded into a Hamming cube $H^{d'}$, where $d' = f_{max} \times d$ represents the number of dimensions for the Hamming cube, f_{max} is the highest value of the fuzzy terms in φ, and $d = \|P\|$. This procedure transforms each item i into a binary vector z_i. The LSH algorithm is described as follows [15]: choose l subsets Q_1,...,Q_j,...,Q_h of $\{$ 1,...,d' $\}$; let $z_{|Q_j}$ denote the projection of vector z on the coordinate set Q_j; the hash function is defined as $g_j(z) = z_{|Q_j}$. The locality sensitive hash function maps the binary representation of the item i

into the bucket $g_j(z_i)$. This pre-processing achieves a localized distribution of the item-set I into the hash buckets.

After inserting the items into the hash buckets, Yoda can retrieve the N nearest items to V_k by visiting the hash buckets one-by-one until the N required items are found. We define a fuzzy distance measure to determine the optimum order of visiting the hash buckets based on their distance from V_k, starting from the closest bucket. Here, we formally define our fuzzy distance measure.

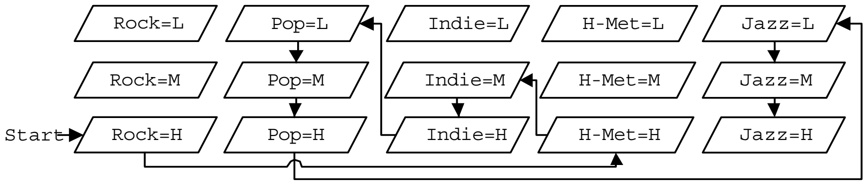


Fig. 2. Filtering Mechanism - Example Schematic Diagram

Definition Set L of all pairs of the properties and their corresponding fuzzy values that need to be searched is defined as:

$$L = \{(p, \tilde{p}) \mid p \epsilon P, \tilde{p} \geq F_p(k)\}$$

There is a one-to-one relationship between elements of L, e_i, and the hash buckets. To select the next bucket in each step, distance of the bucket centroid β_j from V_k, $D(\beta_j, V_k)$, is measured as:

$$S_k(p) = \begin{cases} fmin\{\tilde{p} \mid \forall(p,\tilde{p}) \in L\} & \exists(p,\tilde{p}) \in L \\ 0 & \text{otherwise} \end{cases}$$

$$D(\beta_j, V_k) = \textstyle\sum_{p \in P}[w(S_k(p)) \times S_k(p) \times \beta_j(p)]$$

where the weight $w(f_l)$ for fuzzy term f_l is defined so that $w(f_l) > d \times w(f_{l-1})$, and $\beta_j(p)$ is the value for property p in bucket centroid β_j. The closest bucket is selected for the next step. During transition between bucket i and bucket $i+1$, L is updated according to the following rule:

$$L \leftarrow L - \{e_i\} \qquad (\forall i > 0)$$

where

$$e_i = \begin{cases} (p, f_{l+1}) & e_{i-1} = (p, f_l < fmax(\varphi)) \text{ } and \text{ } i > 1 \\ (p, fmax\{\tilde{p} \mid \forall(p,\tilde{p}) \in L\}) & \text{otherwise} \end{cases}$$

Figure 2, illustrates an example search path according to FLSH retrieval algorithm. The transition diagram is generated based on the favorite PVs of the cluster k defined as: $F_{Rock}(k) = High$, $F_{Pop}(k) = Low$, $F_{Indie}(k) = Medium$, $F_{Heavy-Metal}(k) = High$, $F_{Jazz}(k) = Low$, where $P = \{Rock, Pop, Indie, Heavy\text{-}Metal, Jazz\}$, and $\varphi = \{Low, Medium, High\}$.

5 Performance Evaluation

We performed an end-to-end simulation to compare accuracy and scalability of Yoda with the basic nearest-neighbor (BNN) method [10]. In this simulation framework, both Yoda and BNN are implemented in Java™, on top of Microsoft ™ Access 2000. We used a Microsoft™ NT 4.0 personal computer with a Pentium™ II 233MHz processor to run our experiments.

5.1 Experimental Methodology

In order to generate synthetic data for evaluation purposes, we propose a parametric algorithm to simulate various benchmarks (see Table 1). First, the benchmark method generates K clusters. Each cluster comprises a browse-list, a list of favorite PVs, and a pattern of navigation as the cluster centroid. Every item in the cluster browse-list is assigned a rating value to be used by BNN. For each cluster, the algorithm then randomly generates S/K users. Each user has a current browse-list, b_u, and an expected browse-list, e_u, that are both constructed around the centroid of cluster k with 30% noise.

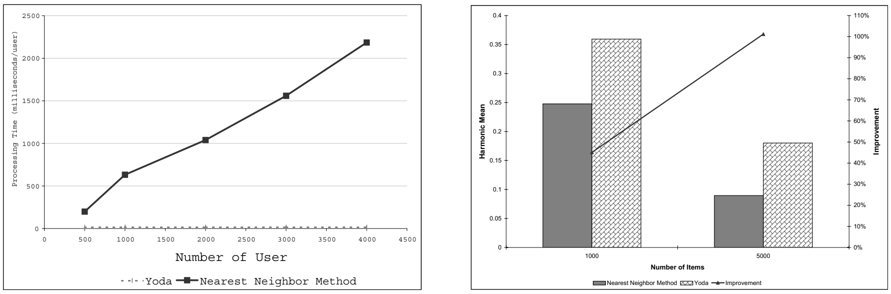
Table 1. Benchmarking Parameters

Parameter	Definition
d	Number of properties ($\|P\|$)
M	Number of items in the item-set ($\|I\|$)
S	Number of user sessions used for analysis
L_{min}	Minimum size of user browse-lists
L_{max}	Maximum size of user browse-lists
Ψ	Number of fuzzy terms ($\|\varphi\|$)
K	Number of clusters
N	The cut-off point

Subsequently, each item in b_u is assigned a rating value based on the original rating value in cluster k. Next, the algorithm generates PVs for each item i based on the favorite PVs of the cluster that has the highest rating for item i, say cluster k'. The higher is the rating of item i in cluster k', the more similar are PVs of i to favorite PVs of cluster k'. The rating values and PVs are represented by fuzzy terms. We use Yoda and BNN to construct wish-list I_u for each user u and compare the wish-list with e_u to evaluate the accuracy of these systems.

5.2 Experimental Results

We conducted several sets of experiments to compare Yoda with BNN. In these experiments, we observed a significant margin of improvement over BNN in



a. Number of Users - Processing Time b. Number of Items - Accuracy

Fig. 3. Impacts of number of users on processing time and number of items on accuracy

matching the user expectations for various settings. Moreover, it is shown that performance of Yoda is independent of the number of users.

The results shown for each set of experiments are averaged over many runs, where each run is executed with different seeds for the random generator functions. The coefficient of variance of these runs is smaller than 5%, which shows our results are independent of the specific run.

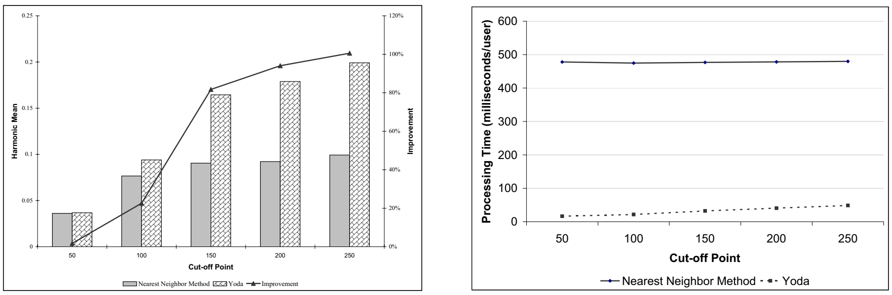
In Figure 3, we compare performances of Yoda and BNN. The benchmark settings of these experiments, K, L_{min}, L_{max}, d, N, and Ψ are fixed at 18, 25, 30, 10, 150, 10, respectively. M in Figure 3.a is 5000 and S in Figure 3.b is 500. X-axis of Figure 3.a is S varying from 500 to 4000, and X-axis of Figure 3.b is M. Y-axis of Figure 3.a is the system processing time for each user, and Y-axis of Figure 3.b depicts the Harmonic mean computed by Equation 1, and improvement computed by Equation 2.

$$\text{Harmonic Mean}(Precision, Recall) = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{1}$$

$$\text{Improvement}(Yoda, BBN) = \frac{(Yoda - BBN)}{BBN} \tag{2}$$

Figure 3.a verifies that Yoda is scalable. As the number of users grow, the system processing time of Yoda remains unchanged while the processing time of BBN increases linearly. This is because Yoda is a model-based recommendation system. Figure 3.b indicates that Yoda always outperforms BNN in accuracy. Although performances of both systems decrease as the number of items grow, the margin of improvement between Yoda and BNN expands. We attribute this improvement to the incorporation of content-based filtering into the Yoda infrastructure.

With Figure 4, we demonstrate impacts of N in accuracy and processing time of the two systems. The benchmark settings of these experiments, K, S, M, d, L_{min}, L_{max}, and Ψ are fixed at 18, 1000, 5000, 5, 25, 30, and 10, respectively. X-axis is N varying from 50 to 250. Y-axis of Figure 4.a depicts the Harmonic



a. Cut-off Point - Accuracy b. Cut-off Point - Processing Time

Fig. 4. Impacts of cut-off point on accuracy and processing time

mean and improvement, and the Y-axis of Figure 4.b is the system processing time.

In Figure 4.a, accuracy of both Yoda and BNN improves as N grows because more items are considered in the wish-list. However, the margin of improvement between Yoda and BNN grows as the cut-off point is increased. Again, applying content-based filtering to retrieve the recommendable items enables Yoda to identify more items similar to the items in the user browse-list. In Figure 4.b, as the cut-off point increases processing time of Yoda grows while processing time of BNN remains unaffected. This observation shows that based on the size of the required wish-list, Yoda searches only a subset of the item-set to generate the wish-lists while with BNN, regardless of the size of the wish-lists, the entire item-set is processed.

6 Conclusions and Future Work

In this paper, we described a recommendation system, termed *Yoda*, that is designed to support large-scale web-based applications requiring highly accurate recommendations in real-time. Our end-to-end experiments show while Yoda scales as the number of users and/or items grow, it achieves up to 120% higher accuracy as compared to the basic nearest-neighbor method. We intend to extend this study in several ways. First, we would like to run more experiments with real data to verify our results and to compare with other approaches. Second, we want to incorporate the content-based filtering mechanism into the *user classification* phase. Finally, our aggregation function is defined in the domain of the original fuzzy logic theory, fuzzy type-I. However, recently Karnik et al. [17] introduced fuzzy type-II to incorporate uncertainty in computation.

Acknowledgments

We are grateful for the assistance given by Jabed Faruque in conducting some experiments and for Amol Ghoting's helpful suggestions in improving the *FLSH* filtering mechanism. This research has been funded in part by NSF grants

EEC-9529152 (IMSC ERC) and ITR-0082826, NASA/JPL contract nr. 961518, DARPA and USAF under agreement nr. F30602-99-1-0524, and unrestricted cash/equipment gifts from NCR, IBM, Intel and SUN.

References

1. Sarwar B., G. Karypis, J. Konstan, and J. Riedl: Analysis of Recommendation Algorithms for E-Commerce, Proceedings of E-Commerce Coneference, 17-20 October, Minneapolis, Minnesota, 2000. 419
2. Fagin R.: Combining Fuzzy Information from Multiple Systems, Proceedings of Fifteenth ACM Symposyum on Principles of Database Systems, Montreal, pp. 216-226, 1996. 420, 424, 425
3. Shahabi C., and Y. Chen: Efficient Support of Soft Query in Image Retrieval Systems, Proceedings of SSGRR 2000 Computer and eBusiness Conference, Rome, Italy, Aguest, 2000.
4. Balabanovi M., and Y. Shoham: Fab, content-based, collaborative recommendation, Communications of the ACM, Vol 40(3), pp. 66-72, 1997. 421
5. Resnick P., N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl: GroupLens, An Open Architecture for Collaborative Filtering of Netnews, Proceedings of ACM conference on Cumputer-Supported Cooperative Work, Chapel Hill, NC, pp. 175-186, 1994. 419, 420
6. Sarwar B., G. Karypis, J. Konstan, and J.Riedl: Analysis of Recommendation Algorithms for e-Commerce, Proceedings of ACM e-Commerce 2000 Conference, Minneapolis, MN, 2000. 421
7. Good N., J. Schafer, J. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl: Combining Collaborative Filtering with Personal Agents for Better Recommendations, Proceedings of the 1999 Conference of the American Association of Artifical Intelligence, pp. 439-446, 1999. 421
8. Kitts B., David Freed, and Martin Vrieze: Cross-sell, a fast promotion-tunable customer-item recommendation method based on conditionally independent probabilities, Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, pp. 437-446, August, 2000. 419, 421
9. Breese J., D. Heckerman, and C. Kadie: Empirical Analysis of Predictive Algorithms for Collaborative Filtering, Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison, WI, pp. 43-52, July, 1998. 419, 421
10. Sarwar B., G. Karypis, J. Konstan, and J. Riedl: Application of Dimensionality Reduction in Recommender System – A Case Study, ACM WebKDD 2000 Web Mining for e-Commerce Workshop, 2000. 421, 428
11. Konstan, J., B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl: Applying Collaborative Filtering to Usenet News, Communications of the ACM (40) 3, 1997. 421
12. Shahabi C., A. M. Zarkesh, J. Adibi, and V. Shah: Knowledge Discovery from Users Web Page Navigation, Proceedings of the IEEE RIDE97 Workshop, April, 1997. 421, 422
13. Shahabi C., F. Banaei-Kashani, J. Faruque, and A. Faisal: Feature Matrices: A Model for Efficient and Anonymous Web Usage Mining , EC-Web 2001, Germany, September 2001. 419, 422, 423

14. Shahabi C., and Y. Chen: A Unified FrameWork to Incorporate Soft Query into Image Retrieval Systems , International Conference on Enterprise Information Systems, Setubal, Portugal, July 2001. 423, 425
15. Gionis A., P. Indyk, and R. Motwani: Similarity search in high dimensions via hashing, Proceedings of the 25th International Conference on Very Large Databases, Edinburgh, Scotland, 1999. 420, 426
16. Weber R., H. Schek and S. Blott: A quantitative analysis and performance study for Similarity Search Methods in High Dimensional Spaces, Proceedings of the 24th International Conference on Very Large Data bases, 1999. 426
17. Karnik N., and J. Mendel: Operations on Type-2 Fuzzy Sets, International Journal on Fuzzy Sets and Systems, 2000. 430

The Use of Machine-Generated Ontologies in Dynamic Information Seeking

Giovanni Modica[1], Avigdor Gal[2], and Hasan M. Jamil[1]

[1] Mississippi State University, Mississippi State University MS 39762, USA
{gmodica, jamil}@cs.msstate.edu
http://www.cs.msstate.edu/~{gmodica,jamil}

[2] Technion, Israel Institute of Technology, Technion City, Haifa 32000, Israel
and Rutgers University, Piscataway, New Jersey 08854, USA
avigal@ie.technion.ac.il
http://ie.technion.ac.il/~avigal

Abstract. *Information seeking* is the process in which human beings recourse to information resources in order to increase their level of knowledge with respect to their goals. In this paper we offer a methodology for automating the evolution of ontologies and share the results of our experiments in supporting a user in seeking information using interactive systems. The main conclusion of our experiments is that if one narrows down the scope of the domain, ontologies can be extracted with a very high level of precision (more than 90% in some cases). The paper is a step in providing theoretical, as well as practical, foundation for automatic ontology generation. It is our belief that such a process would allow the creation of flexible tools to manage metadata, either as an aid to a designer or as an independent system ("smart agent") for time critical missions.

1 Introduction and Motivation

Information seeking is the process in which human beings recourse to information resources in order to increase their level of knowledge with respect to their goals. Dating years back, information seeking has affected the way modern libraries operate (using tools such as catalogs, classifications, and indexing) and perpetrated the World Wide Web in the form of search engines. While the basic concept of information seeking remains unchanged, a growing need of automation of the process has called for innovative tools to propagate some of the tasks involved in information seeking to the machine level. Therefore, databases are widely used for the efficient storage and retrieval of information. Also, techniques from the area of Information Retrieval [21] were refined over the years to predict the relevance of information to a person's needs and to identify appropriate information for a person to interact with. Finally, the use of computer-based ontologies [25] was suggested to classify the available information based on some

© Springer-Verlag Berlin Heidelberg 2001

natural classification scheme that would allow a more focused information seeking.[1]

Most Internet portals (including Yahoo! and OpenDirectory) use "cybrerians" to maintain Internet Directories. Common practice nowadays assume that once ontologies are created, computer-supported tools can utilize them as part of the information seeking process. The next natural step in then to let the machine generate the ontologies. One may consider two incentives in doing so. The first is rooted in the initial creation of ontologies, which is a tedious, time-consuming process. The second incentive is rooted in the rapid evolution of ontologies. If ontologies are managed manually, any change to them requires human intervention. This can bring to a halt an electronic process in the absence of constant human support. While the latter attracted a little attention in previous years, it has become a major sticking point with the introduction of eCommerce and electronic exchange markets, a rapidly changing environment in which virtual manufacturers, retailers, and consumers join in to perform activities in cyberspace.

It is the evolution of ontologies which we offer to automate. In particular, we suggest utilizing ontologies in supporting a user in seeking information using interactive systems. As an example, consider a researcher who is interested in renting a car to attend her favorite conference (*e.g.*, CoopIS) in her favorite city (say Trento, Italy). Using Web services, the researcher attempts at comparing available rates from many different car rental companies, in order to reach an educated decision in obtaining her goal. Alas, this process of information seeking is tedious as well as frustrating. Information has to be typed in over and over again, and in most cases a manual comparison of terms and conditions is needed in evaluating the outcomes. An alternative exists in the form of car rental portals (*e.g.*, Travelocity.com). However, as most general-purpose tools, such portals cater to popular needs, and therefore may only offer a limited set of options (such as the cheapest car available). Therefore, if our researcher is interested in a deal which offers no mileage constraints (since our researcher will attend another conference, say in Rome) and no constraints on cross-border usage (for the purpose of vacating at Switzerland once the conference is over), she has to resort to manual search of terms and conditions.

Figure 1 outlines the various stages of ontology creation and adaptation, as suggested in this paper. An initial ontology is created, using either extraction tools or an existing ontology. Equipped with the ontology, the user performs an information seeking session, in which the machine captures the inserted data and matches it with the ontology. This step is followed by an iterative process, in which new information seeking sessions are performed automatically. Each such session requires the fine tuning of the available ontology to the one currently in use, to be followed by an automated information seeking. The results become available to the user and additional feedback is used to enhance the existing ontology and to improve the system's capabilities for the next session. We term the initial session a *training session*, since this is the session in which the machine

[1] For example, ontology.org is an independent industry and research forum, formed in 1998, that is focused upon the application of ontologies in Internet commerce.

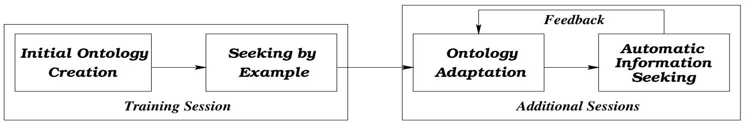


Fig. 1. The process of ontology adaptation

learns the data needs of the user. There is also a continuous learning process, which enables the machine to improve the ontology with each additional session.

We shall use as a running case study the Web sites of three car rental companies, namely Avis, Hertz and Alamo. We shall use the former as the training Web site for creating the initial ontology and the latter to perform ontology adaptation. In our experiments we extract ontologies from HTML documents. We recognize the fact that XML may serve as a better candidate for ontology exploration. In fact, while one can **exarch** terms and structure from XML documents, one has to **mine** for ontologies in HTML documents. However, current trends in deploying XML as part of the organization data management scheme suggest that while XML may be used for B2B communication, and to some extent as a storage mechanism, interactive sessions still use HTML. Therefore, it is possible that XML data on the server side is "translated" into HTML before being shipped out to the client. It is also worth noting that once an ontology is extracted (from either XML or HTML documents), the process of ontology adaptation remains unchanged.

1.1 Related Work

The problem we tackle in this paper falls into the category of *semantic heterogeneity*, which is well documented in the literature. The area of information science has an extensive body of literature and practice on ontology construction using tools such as thesauri and on terminology rationalization and matching of different ontologies [3,22,26,28]. In the area of databases and information systems many models were suggested to support the process of *semantic reconciliation*, including the SIMS project [4], SCOPES [18], dynamic classificational ontologies [14], COIN [16], and CoopWARE [13], to name a few. What is common to these solutions is their reliance on the designer's intervention, rather than supporting a fully automatic semantic reconciliation.[2] However, redesign and re-implementation of metadata can incur tremendous cost. Therefore, automatic reconciliation becomes a must in such an a environment.

Database research has extensive literature on data integration, including [8,2], and [15], yet there is little impact of this research on the state-of-the-art in commercial systems. We believe this chasm can be attributed to the fact that most of these approaches rely on semantic reconciliation to be resolved first (probably manually), before attending to the more "technical" aspects of the integration.

[2] The slogan of OpenDirectory, "Humans do it better," reflects this approach.

However, researchers and practitioners alike are coming to realize that there can be no solution to the delivery of integrated information unless one tackles head-on the semantic heterogeneity problem [19]. This research works towards this goal.

Some research was devoted to automatic schema analysis and integration (*e.g.*, [23], [17], and [9]). In [23], the analysis is based on a hand-crafted attribute hierarchy, which we avoid. The work of [9] and [17] are similar in that they analyze a schema, given in an abstract form of a graph, using formal methods of graph analysis. The tools and methodologies suggested in [9], when applied to schema integration are "not sufficient and must be enriched with semantic consideration, such as the interpretation of terms within an application domain in order to correctly compare elements." Our experiments, as shown in this paper, show that it is possible to automatically (and correctly) derive matchings, without reverting to manual interpretation. In [17], it is shown that the process of finding an optimal typing for semi-structured data is NP-hard. Therefore, a method is presented based on heuristics to approximately type a large collection of semi-structured data. No extension of the method to deal with schemata matching is given in [17].

Like many before us, we attempt to perform semantic reconciliation using syntactic comparisons. However, we also enhance our model to include a measure of accuracy, which becomes a powerful tool whenever automated reasoning is involved. The provision of a measure of accuracy allows a user to determine her own tolerance to imprecision and to instruct the system to request for help once imprecision becomes too great. As our experiments demonstrate, if one narrows down the scope of the domain, ontologies can be extracted with a very high level of accuracy.

The rest of the paper is organized as follows. Section 2 outlines our methodology in creating ontologies from dynamic Web pages. Ontology adaptation is discussed in Section 3. Section 4 provides a brief overview of the system's architecture. Finally, some experiment results and future research directions are discussed in Section 5.

2 Ontology Creation

In this paper we develop a methodology for ontology creation in which the user plays an important role, yet with minimum effort. We hope to make this methodology into a fully automated model for ontology creation at a later stage. The methodology involves two phases, namely the *training phase* and the *adaptation phase*. In the training phase we build an initial ontology in which a user's data needs are encoded. The adaptation phase involve an iterative mergence of closely related ontologies with the current ontology at hand. During each iteration, the current ontology is refined and generalized. The refined ontology at any stage can be used to query the Web sites and gather information in a seamless way.

This section provides an overview of the process of ontology creation. Section 2.1 presents the underlying ontological structures we have utilized in this process, based on [6,7]. The creation of the initial ontology is given in Section 2.2.

2.1 The Ontological Structures

In our ontological analysis we have used the work of Bunge [6,7]. We have adopted a conceptual modeling approach rather than a knowledge representation approach (in the AI sense). While the latter requires a complete reflection of the modeled reality for an unspecified intelligent task, to be performed by a computerized system in the future [5], the former requires the minimal set of structures to perform a given task (in our case, assisting a user in handling dynamic information seeking). Therefore, we have chosen a subset of the ontological constructs provided by Bunge for our work and have added a new construct, we term *precedence*, for posing temporal constraints. We recognize the limited capabilities of HTML (and to that effect, also XML) to represent rich ontological constructs, and therefore we have had to eliminate many important constructs (*e.g.*, the class structure), simply because they cannot be realistically extracted from the content of Web pages.

We shall now provide a brief description of the ontological constructs to be used in this paper.

Terms (things): We extract a set of terms[3] from a Web page, each term is associated with one or more form entries. The term is taken from the labeling of the entry and the match is done based on the proximity of the label and the entry. For example, some of the terms we have extracted from the Avis reservation page are `Pick-Up Location Code`, `Pick-Up Date`, `Pick-Up Time`, `Return Date`, `Return Time`, and `Return Location Code`. The usefulness of this construct is further exemplified in Section 3.

Values: Based on Bunge [6], an attribute is a mapping of things and value-sets into specific statements. Therefore, we can consider a combination of a label and its associated data entry (value) to be an attribute. In certain cases, the value-sets to be associated with a term are constrained using drop lists, check boxes, and radio buttons. For example, the term `Pick-Up Date` is associated with two value-sets, the first is $\{Day, 1, 2, \ldots, 31\}$ and the other is $\{January, February, \ldots, December\}$. Clearly, the former is associated with the date of the month and the latter is associated with the month. Whenever constrained value-sets are available, we can enhance our knowledge of the domain, since such constraints become valuable when comparing two terms that do not exactly match through their labels. For example, the term being used by Alamo for `Return Date` is `Dropoff Date`. Although the terms do

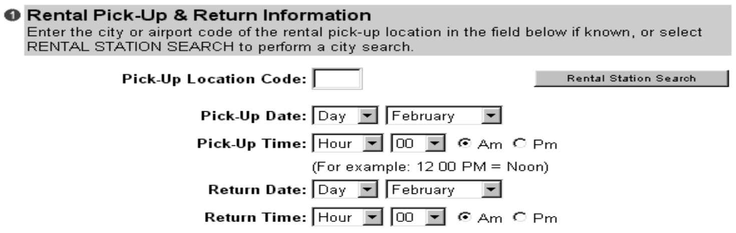
[3] The choice of words to describe ontological structures in Bunge's work had to be general enough to cover any application. We feel that the use of *thing*, which may be reasonable in a general framework, can be replaced with a more concrete description in this application.

not match, and the terms `Return` and `Dropoff` are not synonyms (dropoff is not even considered a word in English, according to Oxford dictionary [1]), our algorithm was able to match these terms using the value-sets, since the term `Dropoff Date` has a value-set of $\{(Select), 1, 2, \ldots, 31\}$.

It is our belief that designers would prefer constraining field domains as much as possible, to minimize the effort in writing exception modules. Therefore, it is unlikely that a field with a dropdown list in one form will be designed as a text field in another form. In the case of a small-sized domain, alternative designs may exist (*e.g.*, AM/PM may be represented as either a dropdown list or radio buttons). Since our algorithm extracts domains and represent them in a unified manner, the end result will remain the same, whether the designer use dropdown list or radio buttons.

Composition: We differentiate simple terms from composite terms. A composite term is composed of other terms (either simple or composite). In the Avis reservation Web page, all of the terms mentioned above are grouped under `Rental Pick-Up & Return Information`. It is worth noting that these terms are, in themselves, composite terms. For example, `Pick-Up Time` is a group of three entries, one for the hour, the other for the minutes, and the third for either AM or PM. In this case, since the entries themselves are nameless, we assign the terms within this group using the group name (*e.g.*, `Pick-Up Time 1`, `Pick-Up Time 2`, etc.) Another composite term in the same Web page is titled `Airline Information` (with the terms `*Airline Name` and `*Flight #` – the * represents an optional field). The algorithm in Section 3 makes use of composition to overcome granularity differences. It is worth noting that there is a rich body of literature on mereology (*e.g.*, [24,27]). However, the minimal support of ontological structures in HTML render these differences immaterial in this framework.

Precedence: The last construct we have considered is the precedence relationship among composite terms. In any interactive process, the order in which one provides the data may be of importance. In particular, data given in an earlier stage may restrict the number of options that are available for a later entry. For example, the Avis Web site determines the car groups available for a given session, using the information regarding the pick-up location and the pick-up time. Therefore, once those entries are filled in, the information is sent back to the server and the next form is brought up. Such precedence relationships can usually be identified by the activation of a script, such as (but not limited to) the one associated with a SUBMIT button. In any such case, we compose all simple and composite terms that are separated by script execution and assign a precedence relationship accordingly. It is worth noting that the precedence construct rarely appears as part of the basic ontology constructs. This can be attributed to the view of ontologies as static entities whose existence is independent of temporal constraints. We anticipate that contemporary applications, such as the one presented in this paper, will need to embed temporal reasoning in ontology construction.


Rental Pick-Up & Return Information
Enter the city or airport code of the rental pick-up location in the field below if known, or select RENTAL STATION SEARCH to perform a city search.

Pick-Up Location Code:
Rental Station Search
Pick-Up Date: Day February
Pick-Up Time: Hour 00 Am Pm
(For example: 12 00 PM = Noon)
Return Date: Day February
Return Time: Hour 00 Am Pm

Fig. 2. Rental Pick-Up and Return Information on Avis Web site

2.2 Target Ontology

Before we describe our ontology extraction process, we introduce two quality metrics, namely *recall* and *precision*, exploited for effective ontology creation through dynamic ontology adaptation.

Metrics Following common IR practice for retrieval effectiveness (*e.g.*, [11]), We shall use the following two metrics for performance analysis. The first is the *recall* (completeness) metric, defined as the ratio of relevant terms retrieved for a given ontology over the number of relevant terms in that ontology. The denominator is taken as the number of fields to be filled during the reservation process. A subjective analysis recovers the numerator. For example, consider Figure 2, which provides the rental pick-up and return information on Avis Web site. The relevant terms are all given to the left of the fields (in this case). However, it is possible that an algorithm would use the statement "*(For example: 12 00 PM = Noon)*" as the term for the line below it (assuming it is a header). This will constitute an irrelevant term.

The second metric is *precision* (soundness), the ratio of the number of relevant terms retrieved over the total number of terms retrieved. In our experiments we shall use the combined measure as suggested in [20]. For a precision value P, a recall value R, and an importance measure b, the combined measure E is computed to be

$$E = 1 - \frac{(1+b^2)PR}{b^2P + R}$$

A low E value indicates a higher combined value of P and R.

Ontology Extraction The extraction process begins with an empty ontology. Once the Web site is accessed by the system browser, the page is parsed into a data structure, called the *DOM tree* (short for *document object model*), which identifies the page elements. This W3C standard for hierarchical data structure is basically a tree that represents the whole HTML page. The DOM tree can be used in a fairly straightforward manner to identify form elements, labels, input elements, *etc.* A DOM tree potentially contains "noise" due to incorrect specification of the source HTML code including difference in order of opening

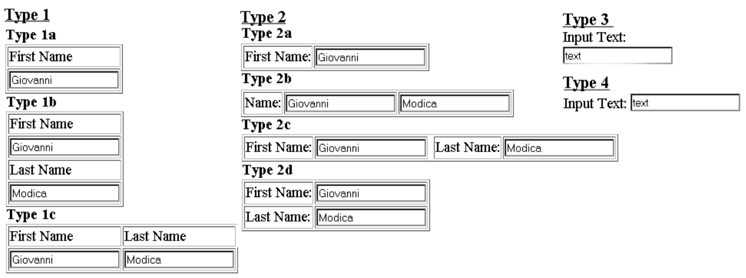


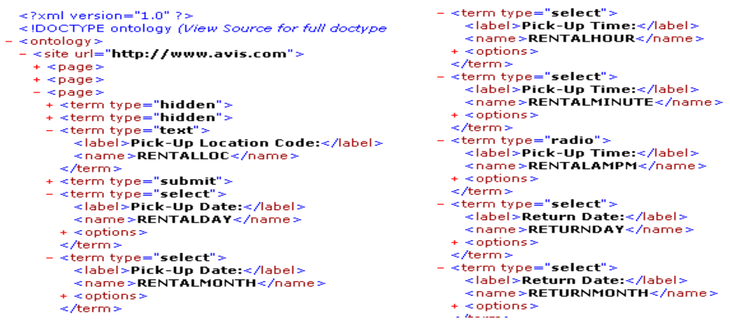
Fig. 3. Possible input layout combinations

and closing tags and missing closing tags. Hence, suitable "cleaning" and filtering is required before achieving an error-free tree. Examples of such cleaning process include elimination of superfluous tags, removal of formatting and scripting tags, *etc.*

The diversity of layout techniques and principles used in Web design complicates the label identification process for input elements even in a well-structured DOM tree. We have used a set of heuristics, learned from a training set of HTML documents, to recognize an HTML page layout. These documents depict all possible table and non-table input layouts we have encountered. Examples of input layout include text and image labels for input elements (or forms), and table type and row type label and input field forms. Figure 3 demonstrates some of these combinations used in the training set. Type 1 and Type 2 show tabular input formats for vertical and lateral layouts respectively. Type 3 and Type 4 show the same for plain (non tabular) input formats. Training the system using this example set resulted in a quite successful label identification process (see Section 3 for details). We attribute this success to the use of the ontological structures presented in Section 2.1. In particular, structures such as terms and values have greatly improved the identification by providing a critical semantic interpretations of source HTML documents. The composition and precedence structures have helped locating terms and associating the labels with the names of the elements by identifying semantic proximity and order of page elements.

The extraction process returns all the identifiable input elements as a set of 5-tuples $\langle u, n, l, t, S \rangle$ where u is the URL of the page (source HTML document), n is the name of the input field, l is the label of the input field, t is the type of the field, and finally, S is the set of values that are listed in a select, radio, or check box fields. The end result of the extraction process is an XML document containing the extracted ontology of the site. The XML document properly identifies the ontological structures discussed in this section to facilitates effective information retrieval at a later stage.

Example of an extracted ontology is shown in Figure 4. The content of this figure can be explained with the help of Figure 2 for Avis Web site, for which the ontology in Figure 4 was generated. In Figure 4, the term `Pick-Up Date` is an example of a successful extraction for the target ontology, while `submit` is not.



```
  <?xml version="1.0" ?>
  <!DOCTYPE ontology (View Source for full doctype
- <ontology>
  - <site url="http://www.avis.com">
    + <page>
    + <page>
    - <page>
      + <term type="hidden">
      + <term type="hidden">
      - <term type="text">
          <label>Pick-Up Location Code:</label>
          <name>RENTALLOC</name>
        </term>
      + <term type="submit">
      - <term type="select">
          <label>Pick-Up Date:</label>
          <name>RENTALDAY</name>
        + <options>
        </term>
      - <term type="select">
          <label>Pick-Up Date:</label>
          <name>RENTALMONTH</name>
        + <options>
        </term>
      - <term type="select">
          <label>Pick-Up Time:</label>
          <name>RENTALHOUR</name>
        + <options>
        </term>
      - <term type="select">
          <label>Pick-Up Time:</label>
          <name>RENTALMINUTE</name>
        + <options>
        </term>
      - <term type="radio">
          <label>Pick-Up Time:</label>
          <name>RENTALAMPM</name>
        + <options>
        </term>
      - <term type="select">
          <label>Return Date:</label>
          <name>RETURNDAY</name>
        + <options>
        </term>
      - <term type="select">
          <label>Return Date:</label>
          <name>RETURNMONTH</name>
        + <options>
```

Fig. 4. Extracted ontology in XML

It is worth noting that in correspondence with the `Pick-Up Date` lateral tabular choice box in Figure 2, we have two label-name pairs in the extracted ontology, i.e., `Pick-Up Date`-*RENTALDAY* and `Pick-Up Date`-*RENTALMONTH*. The latter is extracted just because it was a term in the Web site which will play no role in the extracted ontology (spurious terms). It is also noteworthy that in case the Web site for which the ontology is being extracted has multiple pages, the ontology is created as a combination of information from all the pages the user has actually visited during the training session.

3 Ontology Adaptation

In the adaptation phase, the user suggests browsing other, similar Web sites. Each such site goes through the extraction process, resulting in a *candidate ontology*. The candidate ontology is then merged with the existing ontology (termed *target ontology*). This process refines and generalizes the existing ontology to include more terms, mapped into the existing ontology, to the extent possible. This process is repeated for each new Web site visited in the adaptation phase.

The adaptation effectiveness is measured in terms of the metrics recall and precision presented earlier. Recall measures the percentage of the candidate ontology terms that were successfully mapped to the terms of the target ontology. Precision, on the other hand, is used to quantify the semantic correctness of ontological mapping from candidate ontology to target ontology. This measure is important because syntactic mappings (matching) of labels are not always semantically correct. Together, recall and precision attempts to quantify the adaptation effectiveness of a set of ontologies.

The adaptation of a candidate ontology with the target ontology involves a series of steps, to be discussed shortly. Several heuristics are used in this process to improve performance. For example, while it is possible to consider case sensitivity, we do not use case sensitivity of labels and terms during adaptation process to improve recall measure. Examples of other heuristics include removal of noise characters, hyphenations, etc. The following steps are performed in succession on the two ontologies in the order they are listed.

Ontologies are merged pair-wise and the best match for each existing term in the target ontology is selected. The terms that are not matched or are poorly matched (below a threshold) are usually not selected and discarded. However, a choice is given to the user during the merging process (see discussion in Section 4) to select any unmatched term for the inclusion in the merged ontology through the Form Viewer module. During a query session, use of such terms in the query may result in accessing only the sites (through the Navigation Module discussed in Section 4) that contain these terms. However, unmatched terms may be matched by declaring them to be synonymous through the use of a thesaurus as we explain later in this section and also in section 4.

Textual Matching: In this step all the terms are compared pair-wise and tested for identical textual match (equality test). Usually the recall after this step is very low as labels and terms are unlikely to be named identically although they represent the same term.

Ignorabale Character Removal: Characters such as '*', '/', '-', *etc.* are treated as "noise" and are considered dispensable, and as such are removed from the terms. Hence, after this step, terms such as "*Country" and "country" will be considered identical. The argument here is that such characters do not contribute in creating a meaningful identifier for a database field name.

De-hyphenation: Labels such as "pick-up" and "pick up" are considered identical (*e.g.*, [10]). Hence, hyphens in labels are removed to improve matching. From our experiments, it turned out that by merging the hyphenated words, one yields better recall than replacing hyphens with white space. Hence, we merge hyphenated words by removing the hyphens. For example, "pick-up" will be replaced by "pickup".

Stop Terms Removal: Common terms such as 'a', 'to', 'in', and 'the' are considered *stop terms* (*e.g.*, [12]). The removal of stop terms improve recall and does not adversely affect the precision.

Substring Matching: Labels are matched pair-wise for substring matching. A parametric threshold for term matching is used.[4] The match effectiveness for two terms t_1 and t_2 is defined as the ratio between the number of words in term t_2 that are substrings of terms t_1 and the number of words in term t_2, providing a measure of the semantic similarity of these two terms. If one term contains more of the words of another term, the more similar they are. For example, the match effectiveness of t_1=`Pickup Location` and t_2=`Pick-up location code` can be computed at 66%.

Content Matching: Fields with select, radio, and check box options are processed using their value-sets. A match effectiveness is applied here too, calculated as the number of options in the second term that match (using substring matching) with the options in the first term, divided by the number of options in the second term. For example, suppose that t_1 is a `Return-time`

[4] Usually a 50% match is considered a good measure, and hence we have used this threshold in all the steps described in this section. However, the user can adjust this threshold through the user interface, if desired.

term and t_2 is a `Dropoff-time` term with values such as {10:00am, 10:30am, 11:00am} and {10:00am, 10:15am, 10:30am, 10:45am, 11:00am} respectively. Then, if we inspect each value of t_2 for a match in t_1 (using substring matching technique already described), we will not find a match for values such as 10:15am, 10:45am, and so on. Hence the match effectiveness of t_2 (with respect to t_1) will be calculated as $\frac{3}{5} = 0.60\%$ for this example. The power of content matching can be further highlighted using the case of terms `Dropoff Date` in Alamo and `Return Date` in Avis. These two terms have associated value sets $\{(Select), 1, 2, \ldots, 31\}$ and $\{(Day), 1, 2, \ldots, 31\}$ respectively, and thus their match effectiveness is $\frac{31}{32} = 97\%$, and hence are identified by our method as semantically identical concepts.

Thesaurus Matching: Finally, terms and labels, not matched before, are matched using an ever expanding thesaurus. The thesaurus is constructed from user interactions with the adaptation module. Mismatched terms are presented to the user for manual matching. Every manual match identified by the user is accepted as a synonym and optionally a label is assigned. Each such manual match expands and enriches the thesaurus. This thesaurus is consulted in future matching to improve recall and precision.

4 System Architecture

The modular architecture of the ontology creation and query system is shown in Figure 5. There are three main modules in this system – (i) user interaction module that includes an HTML Parser and an interactive visualization module, (ii) an observer module, consisting of the DOM analyzer, Navigation Module, Form Viewer, HTML Element Viewer, and the HTML Viewer, and (iii) an Ontology Module.

The user interacts with the system through the user interaction module. She accesses a set of Web sites of interest through this module and provides feedback to the system. Each visited site is parsed by the HTML Parser with

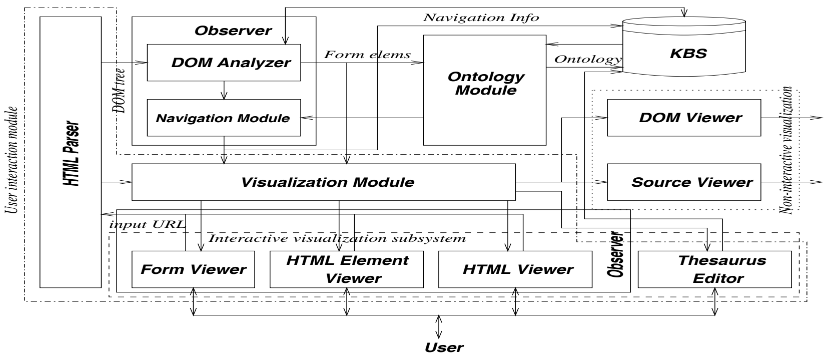


Fig. 5. The system architecture

the help of an HTML/XML parser library to produce a DOM tree that represents the page. During this process, the input page is filtered to remove identifiable "noise" such as formatting tags (*e.g.*, <font>) and scripting tags (*e.g.*, <script>) before it is passed on to the DOM Analyzer. The DOM Analyzer identifies the HTML elements for the Ontology Module. Examples of HTML elements are <a>, <form>, <input>, and <select> elements, and <meta> and <frame> tags. The DOM Analyzer also captures the labels for each element in a form and forwards the information to the Navigation Module, to navigate and query the Web sites when an actual query is submitted by the user in a query session following the ontology extraction.

The observer module gathers information from the user interaction sessions. The information is forwarded to the Ontology Module for creating a target ontology or adapting a candidate ontology to a target ontology. The Ontology Module applies the methodology presented in Section 3 to perform its functions. The Navigation Module stores site-related information for future use. Its role in query processing is to recreate the memorized user navigation pattern and apply site specific terminologies in queries and thus it acts like a wrapper generator.

The Visualization Module acts as a link between the system and the user. It presents a Web browser-like functionality through which the user can interact with the Web sites. The Form and HTML element viewers are used in conjunction with the HTML viewer for this interaction. These interactions cycle through the HTML Parser, the observer and the Visualization Module before they become visible through these viewers again. The DOM Viewer and the Source Viewer are non-interactive in nature and are used for referencing purposes. The system also maintains a user editable thesaurus for effective ontology creation. Once the ontology is created, the thesaurus is automatically updated with new information.

5 Evaluation and Conclusion

In this work we have presented a tool to assist users in dynamic information seeking, using machine-generated ontologies. Our experiments show promise in creating ontologies by extracting concepts and structure from Web pages and matching two somewhat different ontologies using tools from classical IR. Before we conclude, we would like to present some results obtained in our study that substantiate our methodology.

In our experiment, we have taken Avis as the target site for creating the car rental ontology. Figure 6 shows two instances in which we adapt Avis with Hertz (left) and Avis with Alamo (right). The X-axis corresponds to the various steps as described in Section 3. The results in Figure 6 are intuitive and expected, and validates our hypotheses. In the case of Avis and Hertz (left), high initial recall accounts for pairing of terms that potentially contains semantically incorrect matches. Once content and thesaurus matching is applied, these matches are rejected giving rise to diminished recall values hand-in-hand with increasing precision values. Avis and Alamo has a complementary situation where we have

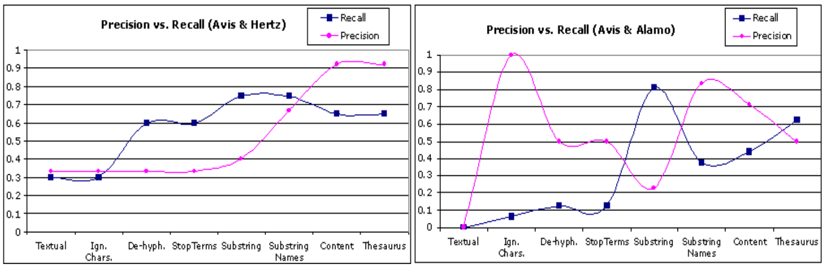


Fig. 6. Precision versus Recall for Avis as target ontology with candidates Hertz (left) and Alamo (right)

high initial precision and low recall that was corrected at the end of the process. This scenario is a result of a very low, but correct (100%), term matches during the initial stages in the merging process. Both metrics have demonstrated good results (more than 50% in both cases with a precision of more than 90% in the case of Hertz).

The metric E, discussed in section 2.2, can be used to demonstrate the gradual improvement of the ontology creation as we iterate through the various steps. Figure 7(left) shows the combined E values for the entire process during ontology creation for Avis-Hertz (left) and Avis-Alamo (right) presented in Figure 6. The plots suggest that the combined measure E gradually approaches zero and thus increases the accuracy of the ontology. Finally, Figure 7(right) shows that an improved thesaurus results in a diminished E value and thus increases the accuracy of the ontology. We have used a b value of 0.5 in all our current experiments. The choice of the parameter $b = 0.5$ in the expression for E indicates the fact that the user is twice as interested in precision than in recall.

The paper is a step in providing theoretical, as well as practical, foundation for automatic ontology generation. It is our belief that such a process would allow the creation of flexible tools to manage metadata, either as an aid to a designer or as an independent system ("smart agent") for time critical missions.

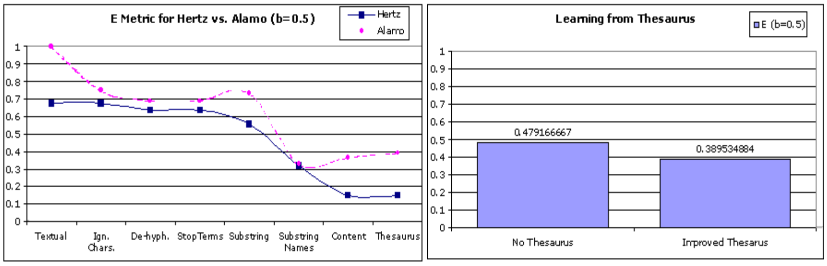


Fig. 7. Combined measure E for the graphs in Figure 6 (left). Effect of a thesaurus (right)

Also, an automatic reconciliation process would allow data management systems to use data even though it is originated from different ontologies.

Acknowledgments

We would like to thank Nicholas Belkin for his assistance.

References

1. *Concise Oxford Dictionary*. Oxford Univ. Press, 8 edition, 1991. 438
2. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The LOREL query language for semistructured data. *International Journal on Digital Libraries*, 1(1), 1997. 435
3. J. Aitchison, A. Gilchrist, and D. Bawden. *Thesaurus construction and use: A practical manual*. Aslib, London, third edition, 1997. 435
4. Y. Arens, C. A. Knoblock, and W. Shen. Query reformulation for dynamic information integration. In G. Wiederhold, editor, *Intelligent Integration of Information*, pages 11–42. Kluwer Academic Publishers, 1996. 435
5. A. Borgida. Knowledge representation, semantic data modelling: What's the difference? In *Proceedings of the 9th International Conference on Entity-Relationship Approach (ER'90)*, pages 1–2, Lausanne, Switzerland, 1990. 437
6. M. Bunge. *Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World*. D. Reidel Publishing Co., Inc., New York, NY, 1977. 437
7. M. Bunge. *Treatise on Basic Philosophy: Vol. 4: Ontology II: A World of Systems*. D. Reidel Publishing Co., Inc., New York, NY, 1979. 437
8. M. J. Carey et al. Towards heterogeneous multimedia information systems: The Garlic approach. In *Proceedings of the RIDE-DOM workshop*, pages 124–131, 1995. 435
9. S. Castano, V. De Antonellis, M. G. Fugini, and B. Pernici. Conceptual schema analysis: Techniques and applications. *ACM Transactions on Database Systems (TODS)*, 23(3):286–332, 1998. 436
10. C. Fox. Lexical analysis and stoplists. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval: Data Structures & Algorithms*, pages 102–130. Prentice Hall, Englewood Cliffs, NJ 07632, 1992. 442
11. W. B. Frakes and R. Baeza-Yates, editors. *Information Retrieval: Data Structures & Algorithms*. Prentice Hall, Englewood Cliffs, NJ 07632, 1992. 439
12. W. Francis and H. Kucera, editors. *Frequency Analysis of English Usage*. Houghton Mifflin, New York, 1982. 442
13. A. Gal. Semantic interoperability in information services: Experiencing with CoopWARE. *SIGMOD Record*, 28(1):68–75, 1999. 435
14. J. Kahng and D. McLeod. Dynamic classification ontologies for discovery in cooperative federated databases. In *Proceedings of the First IFCIS International Conference on Cooperative Information Systems (CoopIS'96)*, pages 26–35, Brussels, Belgium, June 1996. 435
15. T. D. Millstein, A. Y. Levy, and M. Friedman. Query containment for data integration systems. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Dallas, Texas, May 2000. ACM Press. 435

16. A. Moulton, S. E. Madnick, and M. Siegel. Context mediation on Wall Street. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (CoopIS'98)*, pages 271–279, New York City, New York, August 1998. IEEE-CS Press. 435
17. S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In L. M. Haas and A. Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 295–306. ACM Press, 1998. 436
18. A. M. Ouksel and C. F. Naiman. Coordinating context building in heterogeneous information systems. *Journal of Intelligent Information Systems (JIIS)*, 3(2):151–183, April 1994. 435
19. A. M. Ouksel and A. P. Sheth. Semantic interoperability in global information systems: A brief introduction to the research area and the special section. *SIGMOD Record*, 28(1):5–12, March 1999. 436
20. C. J. Van Rijsbergen, editor. *Information Retrieval*. Butterworths, London, 1979. 439
21. G. Salton and M. McGill. *Modern Information Retrieval*. McGraw-Hill, New York, 1983. 433
22. P. L. Schuyler, W. T. Hole, and M. S. Tuttle. The UMLS (Unified Medical Language System) metathesaurus: representing different views of biomedical concepts. *Bulletin of the Medical Library Association*, 81:217–222, 1993. 435
23. A. P. Sheth, S. K. Gala, and S. B. Navathe. On automatic reasoning for schema integration. *Intenational Journal on Intelligent Cooperative Information Systems (IJICIS)*, 2(1):23–50, June 1993. 436
24. P. Simon. *Parts: A Study in Ontology*. Clarendon Press, New York, NY, 1987. 438
25. H. Smith and K. Poulter. Share the ontology in XML-based trading architectures. *Communications of the ACM*, 42(3):110–111, 1999. 433
26. D. Soergel. *Organizing information : principles of data base and retrieval systems*. Academic Press, Orlando, FA, 1985. 435
27. A. Varzi. On the boundary between mereology and topology. In R. Casati, B. Smith, and G. White, editors, *Philosophy and the Cognitive Sciences*. Hoelder-Pichler-Tempsky, Vienna, Austria, 1994. 438
28. B. C. Vickery. *Faceted classification schemes*. Graduate School of Library Service, Rutgers, the State University, New Brunswick, N. J., 1966. 435

Author Index